

Contents

WebSphere Developer Tools	1
Overview: IBM WebSphere Application Server Developer Tools for Eclipse, Version 8.5.5.3	
Developer Tools known restrictions	7
Notices	9
Privacy Policy Considerations	12
Tutorials	13
Java tutorials	14
Create a Hello World Java application	15
OSGi tutorials	16
OSGi EJB tutorial	17
Lesson 1: Create an OSGi project with EJB support	18
Lesson 2: Develop an EJB	21
Lesson 3: Develop a web bundle that accesses the EJB	25
Lesson 4: Test the application	33
Develop a simple OSGi application	35
Introduction: Develop a simple OSGi application	36
Lesson 1: Create the bundle and application	38
Lesson 2: Develop the business logic	40
Lesson 3: Create the blueprint configuration file	43
Lesson 4: Create a servlet that accesses an OSGi service	45
Lesson 5: Deploy the OSGi Counter application	48
Lesson 6: Add dependency injection within a bundle	49
Lesson 7: Add dependency injection between bundles	53
Summary	58
Web tutorials	59
Create a loan payment calculator with Dojo	60
Introduction: Create a loan payment calculator with Dojo	61
Lesson 1: Create a Dojo-enabled web project	63
Lesson 2: Create a custom Dojo widget	66
Lesson 3: Add the custom Dojo widget to a web page	72
Lesson 4: Add a pie chart using the Dojo charting API (optional)	77
Lesson 5: Create a Dojo custom build (optional)	81
Lesson 6: Debugging your Dojo application with Firebug (optional)	82
Summary: Create a loan payment calculator with Dojo	84
Web services tutorials	85
Creating a JAX-WS web service and an EJB skeleton from a WSDL file	86
Lesson 1.1: Set up the workspace and create the required projects	87
Lesson 1.2: Download and validate the WSDL file	90
Lesson 1.3: Create the web service	91
Lesson 1.4: Implement the temperature conversion methods	93
Lesson 1.5: Validate the web service traffic WS-I compliance	94
Summary	96
Creating a JAX-RS web service	97
Lesson 1: Creating a server and web project	98
Lesson 2: Creating and testing the web service	100
Creating a secured JAX-WS web service and client from a WSDL file	102

Lesson 1: Creating a server and web project.....	103
Lesson 2: Creating the web service.....	107
Lesson 3: Creating the web service client.....	110
Lesson 4: Attach the WS-I RSP policy set to the web service.....	113
Lesson 5: Secure the web service client with the WS-I RSP policy set.....	114
Summary.....	116
Samples.....	117
Web services.....	118
WebSphere JAX-WS Web service temperature conversion sample.....	118
WebSphere JAX-WS Web service temperature conversion setup instructions.....	119
OSGi.....	120
OSGi Counter Service.....	120
OSGi Blog.....	122
OSGi Blog setup.....	124
OSGi Blog details.....	125
OSGi Hello World.....	126
OSGi EJB temperature converter.....	128
Setup instructions.....	129
OSGi Calculator.....	131
Installing WebSphere Application Server Developer Tools for Eclipse.....	132
System requirements for WebSphere Application Server Developer Tools for Eclipse.....	133
Installing and updating into an existing Eclipse workbench.....	135
Creating a runtime environment with an optional IBM SDK.....	142
Converting an unlicensed version of WebSphere Application Server Developer Tools for Eclipse into a licensed version.....	144
Known problems and limitations for WebSphere Application Server Developer Tools for Eclipse, Version 8.5.5.....	145
Developing.....	148
Developing enterprise applications.....	148
Learn about enterprise applications.....	149
Developing Java EE Applications.....	150
Learn about Java EE Applications.....	151
Java EE: Overview.....	152
Support for Java EE 7.....	154
Tools for Java EE development.....	155
Java EE perspective.....	157
Enterprise explorer view in the Java EE perspective.....	159
Project facets.....	162
Selecting working sets.....	163
Setting Java EE preferences.....	165
Selecting default project structures.....	167
Setting Java compiler compliance.....	169
Creating and configuring Java EE projects using wizards.....	170
Creating enterprise application projects.....	171
Creating application client modules.....	172

Creating EJB modules using wizards.....	173
Creating web modules.....	175
Using loose class path web libraries support.....	177
Creating and configuring resource references for Web 2.5, Web 3.0, and Web 3.1.....	181
Creating web fragment projects.....	184
Creating connector modules.....	186
Deleting Java EE modules.....	187
Importing JAR, WAR, EJB JAR, client application JAR, and RAR files...	189
Exporting JAR, WAR, EJB JAR, client application JAR, and RAR files...	191
Customizing the Liberty class path container.....	192
Configuring Bean Validation.....	193
Creating and configuring Java EE modules using annotations.....	194
Java EE with annotations overview.....	196
Scope and placement of annotations.....	198
Annotations view.....	200
Defining your own annotations.....	202
Adding annotations.....	204
Adding annotations using the Annotations view.....	205
Adding annotations manually.....	208
Editing annotations.....	209
Editing annotations using the Annotations view.....	210
Editing annotations manually.....	211
Removing annotations.....	213
Removing annotations using the Annotations view.....	214
Removing annotations manually.....	215
Viewing overridden annotations.....	216
Excluding files from annotation scanning.....	217
Types of annotations.....	218
Defining Java EE applications.....	219
Using the application deployment descriptor editor.....	220
Application deployment descriptor editor.....	221
Generating deployment descriptors.....	223
Generating WebSphere extensions and bindings deployment descriptors.....	224
Specifying dependent JAR files or modules.....	225
Adding modules to an enterprise application.....	226
Exporting and importing binary modules.....	227
Java EE deployment assembly.....	228
Editing the Java EE deployment assembly page.....	230
Enterprise application security.....	234
Annotations to secure Java EE applications.....	235
Setting permissions in the Java EE Application Permission Editor.....	237
Validating code in enterprise applications.....	238
Validator tuning.....	240
Validating Java EE projects.....	242
Manually validating code.....	243

Common validation errors and solutions	244
Disabling validation	247
Selecting code validators	248
Overriding global validation preferences	249
Developing EJB 3.x applications	250
EJB 3.x overview	251
EJB modules	252
Creating enterprise beans	253
Enterprise beans overview	254
Creating enterprise beans using wizards	256
Creating a stateless session bean using wizards	257
Creating a session bean with JPA entities	259
Creating a stateful session bean using wizards	261
Creating an EJB timer by using wizards	263
Creating a message-driven bean using wizards	265
Packaging EJB content in web application archive (WAR) modules	267
Deleting EJB 3.x beans	269
Creating enterprise beans using annotations	271
Creating a stateless session bean using annotations	273
Creating a stateful session bean using annotations	276
Creating a message-driven bean using annotations	279
Common annotations	281
EJB annotations	282
Importing class files	283
Editing EJB 3.x applications	284
Using as-you-type validation	285
Content assist and EJB 3.x	286
Deleting EJB 3.x beans	287
Refactoring EJB 3.x Java elements	287
Enterprise application security	288
EJB Security	288
Testing EJB 3.x applications	290
Testing EJB 3.1 applications	291
Developing Java Persistence API (JPA) applications	292
Learn about JPA applications	293
JPA architecture	294
Entity manager	296
JPA query language	298
Creating JPA projects	299
Adding JPA support to a project	300
Converting a Java project to a JPA project	301
Changing a JPA 1.0 project to JPA 2.0	302
Creating JPA entity beans	303
Creating a JPA entity bean using a wizard	304
Creating a JPA entity bean by adding persistence to a POJO	306
Creating JPA entity beans in a JPA project from database tables	307
Creating JPA entity beans in a JPA-enabled web project from existing database tables	308

Generating database tables from entity beans (top-down mapping)	309
Generating a data definition language file from a JPA project	310
Configuring JPA entity beans	311
Configuring the primary key	312
Configuring named queries	313
Configuring relationships	314
Configuring concurrency control	315
Configuring advanced options	316
Adding a primary key to the JPA entity	317
ID annotation	318
IDclass annotation	319
Embedded ID annotation	320
Synchronizing persistent classes in the persistence.xml file	321
Working with persistence units (persistence.xml)	322
Modifying the persistence unit details	323
Adding or overriding entity mappings in the orm.xml file	324
Developing WebSocket applications	325
Creating WebSocket endpoints	326
Deploying WebSocket applications	327
Developing applications that use Contexts and Dependency Injection (CDI)	328
Learn about Contexts and Dependency Injection (CDI)	329
Implied values in contexts and dependency injection annotations	330
Editing and validating your contexts and dependency injection projects	331
Developing OSGi applications	332
Learn about OSGi applications	333
OSGi overview	337
The OSGi specification	339
OSGi Blueprint Container Specification	340
OSGi Blueprint XML files	342
OSGi architecture	343
OSGi bundles	345
OSGi applications	348
OSGi fragments	351
OSGi composite bundles	353
Tools for OSGi application development	356
Installing the server	357
Creating OSGi projects	360
OSGi project facets	361
Creating OSGi bundle projects	363
OSGi bundles	365
Creating web enabled OSGi bundle projects	365
Creating JPA enabled OSGi bundle projects	366
Creating JAX-RS enabled OSGi bundle projects	367
Editing bundle manifest files	368
Declaring dependencies outside of the workspace	369
Declaring dependencies outside of the workspace that resolve at run time	370

Specifying version range	371
Exploring bundle dependencies	373
Creating blueprint XML files	374
OSGi Blueprint XML files	380
Creating blueprint binding XML files	380
Creating fragment projects	381
OSGi fragments	382
Editing fragment manifest files	382
Creating composite bundle projects	383
OSGi composite bundles	384
Editing composite bundle manifest files	384
Adding bundles to composite bundles	385
Creating OSGi application projects	386
Editing application manifest files	388
Application manifest files	389
Viewing deployment manifest files	391
Deployment manifest files	392
Adding bundles to OSGi application projects	394
Adding composite bundles to OSGi application projects	396
Adding bundle fragments to OSGi application projects	398
Importing and configuring OSGi applications	399
Importing OSGi application projects	400
Importing OSGi bundle projects	401
Importing OSGi composite bundle projects	402
Importing OSGi fragment projects	403
Importing deployment manifest files	404
Exporting OSGi applications	405
Exporting OSGi application projects	406
Exporting OSGi bundle projects	407
Exporting OSGi composite bundle projects	408
Exporting OSGi fragment projects	409
Exporting deployment manifest files	410
Converting existing projects to OSGi projects	411
Adding OSGi support to Maven projects	413
Adding Maven support to OSGi applications	414
Exporting packages from a JAR file in a Maven bundle	415
Accessing data using Java Persistence API (JPA)	416
Accessing data by using JPA inside of the bundle	417
Accessing data using JPA in a different bundle	419
Deploying OSGi applications	422
Deploying OSGi application projects	423
Deploying OSGi composite bundle projects	425
Removing OSGi business level application from WebSphere Application Server 427	
Using independent Java archives (JARs) with OSGi applications	428
Creating an OSGi bundle from a JAR file	429
Including a JAR file in an OSGi application	431

Depending on a JAR file without including it in an OSGi application.....	432
Managing extensions.....	434
Extending applications by selecting composite bundles.....	434
Associating composite bundle extensions with an application.....	435
Exposing EJBs as OSGi services.....	437
Configuring OSGi bundle repositories.....	438
OSGi application samples and tutorials.....	440
Developing Maven projects.....	441
Maven overview.....	442
Tools and projects overview.....	444
Setting up your environment.....	447
Downloading and enabling the Maven repository index.....	448
Setting up the IBM Maven repository.....	449
Configuring dependency POM files that emulate the classpath of specific WebSphere runtime environments.....	450
Installing the server APIs into the Maven repository.....	453
Working with projects.....	456
Creating Maven projects.....	456
Importing projects.....	458
Converting existing projects to Maven.....	460
Setting POM entries for projects that target WebSphere Application Server.....	465
Defining Java EE module dependencies.....	466
Adding dependencies to other modules.....	467
Adding modules to an EAR project.....	469
Adding libraries to the EAR library directory.....	470
Workspace preferences.....	472
Setting Maven preferences.....	472
Accessing the Maven console.....	474
Accessing the Maven Console.....	474
Reference.....	476
Example archetypes.....	476
Developing web applications.....	478
Learn about web applications.....	479
Web development tools.....	481
The web perspective.....	482
Enterprise Explorer view and web development.....	484
Palette view.....	486
Rich Page Editor.....	487
Properties view associated with Rich Page Editor.....	490
Mobile Navigation view.....	491
Mobile browser simulator.....	494
Creating web projects.....	496
Web projects.....	498
Creating web projects for mobile devices.....	501
Creating JPA-enabled web projects.....	502
Creating Web 2.0-enabled web projects.....	503
Installing the WebSphere Application Server Feature Pack for Web 2.0.....	504

Creating Dojo-enabled web projects.....	507
Configuring Dojo project libraries.....	509
Web 2.0 features.....	511
Web project features.....	513
Refactoring Web project content roots.....	517
Enterprise Explorer view and web development.....	518
Creating website styles.....	518
Creating style sheets.....	519
Specifying styles for predefined areas.....	520
Setting the CSS profile.....	521
Creating and importing web pages and resources.....	522
Creating web pages.....	523
Adding elements to a web page from the palette.....	524
Adding Flash objects to a web page.....	525
JavaServer Pages (JSP) technology.....	526
Defining HTML file preferences.....	527
Specifying DOCTYPE.....	528
Rich Page Editor.....	529
Browser requirements for Rich Page Editor.....	529
Embedded browsers for Linux.....	531
Configuring for the WebKit embedded browser.....	532
Configuring for the XULRunner embedded browser.....	533
Setting the Rich Page Editor preferences.....	534
Opening web pages in Rich Page Editor.....	536
Working in the Design and Split views.....	537
Working in the Source view.....	538
Creating web pages in Rich Page Editor.....	541
Creating web pages for mobile devices.....	542
Mobile patterns.....	544
Adding a mobile pattern to an application.....	545
Creating mobile pattern projects.....	546
Adding Dojo framework to a UI Pattern Project.....	548
Adding jQuery framework to a UI Pattern Project.....	549
Adding elements to web pages from the palette.....	550
HTML5 tags in palette.....	551
Properties view associated with Rich Page Editor.....	552
Mobile Navigation view.....	553
Creating servlets.....	553
Servlets.....	554
Creating filter classes.....	555
Creating listener classes.....	556
Importing web archive (WAR) files.....	557
Web archive (WAR) files.....	558
Exporting web archive (WAR) files.....	559
Creating web-based user interfaces.....	560
Creating Web 2.0 applications.....	561
Asynchronous JavaScript and XML (AJAX) overview.....	562

The Dojo Toolkit.....	564
Creating dynamic display.....	565
Creating Dojo applications.....	566
Adding Dojo widgets to web pages.....	567
Creating custom Dojo widgets.....	568
Adding Dojo widgets to existing Java EE web pages.....	569
Editing Dojo widget attributes.....	570
Customizing the properties for Dojo layout widgets.....	571
Creating Dojo classes.....	572
Configuring Dojo project libraries.....	573
Creating Dojo classes.....	573
Establishing a connection between the browser and server-side events (web messaging).....	573
Web messaging.....	574
Rendering data as JSON.....	577
JavaScript Object Notation (JSON4J).....	578
Editing JSON files with the JSON editor.....	579
Setting JSON editor preferences.....	580
Creating a Dojo profiled build (custom build).....	581
Running a Dojo custom build.....	584
Including a Dojo layer file in your web page.....	586
Developing mobile web applications.....	587
Adding Dojo mobile widgets to your mobile web page.....	588
Editing web pages for mobile devices.....	590
Adding tag library support for web projects.....	591
Tag library support for web projects.....	592
Adding support for a component library.....	593
Creating a library project.....	594
Creating a JSP Library Definition (CLD).....	595
Configuring the JSP Library Definition.....	596
Updating a library definition.....	602
Adding translations to a library definition.....	603
Adding support for custom tag libraries.....	605
Creating custom Dojo widgets.....	606
Adding custom JSP tags.....	606
Custom tag libraries.....	608
Adding the Tag Library Descriptor (TLD) file.....	609
Adding a taglib directive to a JSP file.....	610
Specifying the taglib directive.....	611
Editing the properties of a custom tag.....	612
Editing a Web deployment descriptor file.....	613
HTML and JSTL tag libraries.....	614
Debugging web applications.....	621
Debugging web pages with Firebug.....	622
Setting Firebug preferences.....	624
Configuring web applications.....	625
Configuring web applications with the web deployment descriptor editor.....	626

Web 3.0 Modules: The web Deployment Descriptor editor.....	627
Generating web deployment descriptors for web 3.0 projects.....	631
Web 2.5 Modules: The web Deployment Descriptor editor.....	632
Adding annotations to a web application.....	636
Testing and publishing web applications.....	638
Testing mobile applications.....	639
Mobile browser simulator.....	641
Calibrating the mobile browser simulator.....	641
Enabling user agent switching.....	642
Creating Dojo Object Harness (DOH) tests for Dojo test automation.....	644
Configuring DOH test properties.....	645
Creating stand-alone DOH test launch.....	646
Testing Dojo applications.....	647
Testing on a Web Preview Server.....	648
Configuring a web Preview Server.....	649
Adding data using Java Persistence API (JPA).....	654
Editing source code.....	655
File encoding.....	657
Content assist.....	658
Validating source.....	660
Configuring validation preferences.....	661
Editing with snippets.....	662
Adding snippets drawers.....	663
Adding items to snippets drawers.....	664
Editing snippet items.....	665
Deleting or hiding snippet items or drawers.....	666
Configuring structured text editor preferences.....	667
Adding and removing markup language templates.....	668
Adding and removing HTML templates.....	669
Adding and removing JSP templates.....	670
Adding and removing XML templates.....	671
Associating editors with additional files types.....	672
Web application tutorials, samples, and videos.....	673
Developing Web service applications.....	674
Learn about web service applications.....	676
SOAP.....	678
SOAP MTOM.....	680
JAX-WS.....	682
JAXB.....	688
Developing JAX-RS applications.....	690
JAX-RS.....	691
Generating a JAX-RS resource from a WADL file.....	692
wadl2java command.....	693
Generating sample client code for a JAX-RS 2.0 resource.....	694
Tools for web services development.....	696
Web services runtime environments.....	698
IBM WebSphere JAX-WS runtime environment.....	699

Configuring a workspace for web services development.....	701
Setting web services preferences.....	703
WebSphere Web services preferences.....	704
Service policies web services preferences.....	706
Web Service Explorer preferences.....	710
Setting the level of WS-I compliance.....	711
Creating a JAX-WS-enabled WebSphere server.....	713
Creating a JAX-WS enabled web project.....	715
Creating a WebSphere Application Server and Web project.....	717
Creating a Liberty profile server and Web project.....	718
Creating a JMS server.....	719
HTTP and JMS transport methods.....	722
Configuring the IBM JRE to talk to a secured WebSphere Application Server.....	723
Problems working with a secured server using SSL connections.....	725
Importing and creating resources for web services.....	726
Developing Web services and clients.....	727
Methods of creating Web services and clients.....	728
Creating web services and clients by using the web service wizards.....	729
IBM WebSphere JAX-WS runtime environment.....	730
Creating a Java bean skeleton from a WSDL document using the WebSphere JAX-WS runtime environment.....	730
Creating an enterprise bean (EJB) skeleton from a WSDL document using the WebSphere JAX-WS runtime environment.....	734
Creating a web service from a Java bean using the IBM WebSphere JAX- WS runtime environment.....	739
Creating a JAX-RS web service.....	743
Creating Web service clients.....	746
Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment.....	747
Creating web services by using annotations.....	750
Annotations for creating web services overview.....	751
Annotating a Java bean.....	754
Creating a web service from a Java bean with a wizard.....	755
Creating a web service from a Java bean by publishing to a server..	757
Annotating an EJB bean to create a web service.....	758
Creating web service router modules.....	760
Creating a web service from an EJB bean by publishing to a server..	761
Creating a web service from Java and WSDL.....	762
Creating a Web service from an EJB bean and a WSDL file.....	763
Validation of web services annotations.....	764
JAX-WS annotations reference.....	765
Web Services metadata annotations (JSR 181).....	766
JAX-WS 2.0 Annotations (JSR 224).....	774
JAX-WS Common Annotations (JSR 250).....	777
Rules for methods in classes annotated with @WebService.....	779
Creating web services and clients with Ant tasks or command line tools.....	780

Creating IBM WebSphere JAX-WS runtime environment web services and clients using Ant tasks	781
Importing Ant files for your JAX-WS web service	782
Customizing the Ant script to run JAX-WS web service Ant tasks in a command line	783
Creating a Java web service	784
Ant properties files for bottom-up Java web services	786
Ant properties files for top-down Java web services	789
Creating a web service client	791
Ant properties files for web service clients	793
Web services: Editing, assembling, and securing tasks	795
Creating and editing JAX-WS web service handlers	796
Merging web services updates	798
Securing web services	799
Web services security overview	800
Qualities of service for JAX-WS web services and clients	802
Web services policies	803
Web services policy sets that are included with the product	804
Creating a policy set attachment on the client side	809
Attaching a policy set to an endpoint	810
Configuring bindings for policy types	811
Web services bindings	812
Configuring bindings for HTTP transport	813
Configuring bindings for SSL transport	814
Configuring bindings for WS-Reliable Messaging	815
Configuring bindings for WS-Security	816
Specifying keystore settings	818
Creating a policy set attachment on the server side	819
Modifying policy set attachments	820
Editing policy set bindings	821
Importing policy sets	823
Deploying web services	824
Generating JAX-WS deployment descriptors	825
Deploying a web service to a server using the command line tools	826
Creating web service router modules	828
Testing and validating web services	828
Testing web services with the Generic Service Client	829
Service testing overview	831
Recording service tests	833
Service testing guidelines	835
Verifying WSDL syntax compliance for JMS services	838
Recording a service test with the generic service client	840
Recording a service test through a client program	843
Creating a service test from a BPEL model	847
Creating a service test manually	849
Creating a service test for WebSphere MQ	851
Creating an XML call test manually	853

Changing service test generation preferences	855
Editing service tests	856
Web service test editor overview	857
Verifying application behavior	859
Adding equal verification points	860
Adding contain verification points	862
Adding Xpath query verification points	866
Adding attachment verification points	867
Adding XSD verification points	868
Adding elements to a service test	869
Adding a service request	870
Updating a service response from the service	871
Manually adding a response element	873
Editing WSDL security profiles	875
WSDL security editor overview	876
Creating security profiles for WSDL files	878
Using a security policy	880
Implementing a custom security algorithm	882
Adding WS-Addressing to a security configuration	885
Testing asynchronous services	887
Asynchronous service testing overview	888
Creating an asynchronous request structure	890
Adding an asynchronous callback to a service request	892
Creating a reliable messaging call structure	894
Simulating services with stubs	896
Service stub overview	897
Creating a service stub	899
Editing a service stub	900
Deploying service stubs	902
Recording service stub activity in a log file	903
Setting log level for service stubs	905
Comparing service test response contents	906
Service test editor preferences	907
Service test details	909
Service call details	915
XML call details	920
Binary call details	927
Text call details	931
Service message return details	935
Service verification point details	937
Service callback details	940
Service timeout details	941
Service parallel details	942
Service receive details	943
Service stub editor reference	945
Stub operation details	946
Stub case details	948

Stub response details.....	951
Generic Service Client preferences.....	953
Service test editor preferences.....	955
Service message edition preferences.....	956
Service test generation preferences.....	957
Raw transaction data view preferences.....	958
Auto values preferences.....	959
Response time breakdown preferences.....	960
Cookies support preferences.....	961
WSDL Information Preferences.....	962
Web service performance test reports.....	963
Service Performance report.....	964
Web Service Verification Points report.....	967
Generic service client overview.....	969
Configuring the environment for service calls.....	972
Configuring the environment for SOAP security.....	973
Sending service requests with the generic service client.....	974
Generic service client overview.....	975
Creating transport protocol configurations.....	975
Creating an HTTP transport configuration.....	976
Configuring the workbench for NTLMv2 authentication.....	978
Creating a JMS transport configuration.....	979
Creating a WebSphere MQ transport configuration.....	981
Creating Microsoft .NET transport configurations.....	983
Creating SSL configurations.....	986
Sending service requests with WSDL files.....	988
Sending HTTP endpoint requests.....	991
Sending a JMS endpoint request.....	993
Sending a WebSphere MQ endpoint request.....	995
Testing all operations in a WSDL file.....	997
Viewing message content.....	999
Synchronizing a remote WSDL file.....	1001
Adding static XML headers to a service request.....	1003
Opening file attachments.....	1005
Editing WSDL security profiles.....	1006
WSDL security editor overview.....	1006
Creating security profiles for WSDL files.....	1006
Using a security policy.....	1006
Implementing a custom security algorithm.....	1006
Adding WS-Addressing to a security configuration.....	1006
Generic service client reference.....	1006
Generic service client call details.....	1007
Generic service client binary call details.....	1018
Generic service client text call details.....	1022
Generic service client message return details.....	1026
WSDL security editor reference.....	1028
Testing web services with sample JSPs.....	1033

Limitations of web services	1035
Developing Java batch applications	1038
Java batch overview for WebSphere Developer Tools	1039
Installing Java EE batch tools	1040
Job Specification Language editor	1041
Job XML substitution in the JSL editor	1042
Creating a Java batch project with the Java EE Batch Project wizard	1044
Creating a Java batch job with the Batch Job wizard	1046
Adding a batchlet step to a Java batch job	1047
Adding a chunk step to a Java batch job	1049
Submitting a Java batch job	1053
Viewing Java batch job logs for WebSphere Developer Tools	1055
Testing and publishing on a server	1056
Learn about testing and publishing on a server	1057
Overview: Testing and publishing on a server	1058
When the test server requires restarting	1060
Servers view	1063
Migrating projects with a target runtime set to WebSphere Application Server ..	1066
Defining server preferences	1068
Server Tools extension types	1069
Defining the server runtime environments preferences	1070
Defining preferences for WebSphere Application Server	1072
Profile creation for a WebSphere Application Server	1073
Creating, editing, and deleting servers	1074
Creating a server	1075
Creating a WebSphere Application Server	1077
Testing on a Web Preview Server	1082
Testing on a J2EE Preview server	1082
Profile creation for a WebSphere Application Server	1083
Creating a profile on a local WebSphere Application Server V8.5, V8.0, or V7.0.	1083
Configuring servers	1085
Configuring a WebSphere Application Server	1086
Profile creation for a WebSphere Application Server	1088
Setting status updates	1088
Setting the connection to the server	1089
Optimizing starting the server for development	1091
Switching to HPEL mode for logging and tracing on WebSphere Application	
Server V8.0 or later	1093
Keeping the server running after exiting the development environment ..	1095
Changing the hypertext transfer protocol settings	1096
Setting publishing options	1097
Controlling incremental publish	1100
Specifying administrative settings to a secured server	1103
Problems working with a secured server using SSL connections ..	1105
Starting a remote WebSphere Application Server	1105
Advanced WebSphere Application Server configurations	1107

Configuring a WebSphere Application Server with Federal Information Processing Standard enabled	1108
Configuring a Web Preview Server	1110
Managing servers	1110
Starting a server	1111
Starting a remote WebSphere Application Server	1113
Stopping a server	1113
Deactivating and activating your server	1114
Accessing the administrative console	1115
Running administrative script files on a WebSphere Application Server	1116
Debugging Jython script files on WebSphere Application Server	1120
Keeping a WebSphere Application Server running after exiting the development environment	1124
Testing applications on a server	1124
Testing artifacts on a server	1125
Testing on a Java Platform, Enterprise Edition Preview server	1126
Debugging applications on a server	1126
Hot method replace when debugging applications on WebSphere Application Server	1127
Debugging a JSP file on a server	1128
Publishing settings for a WebSphere Application Server	1129
When to wait for the time interval of an automatic publishing to pass on a WebSphere Application Server	1129
Controlling incremental publish	1133
Limitations and troubleshooting tips for Server Tools	1133
General workbench issues	1135
Troubleshooting if a problem is related to the workbench or the application server	1135
Limitations of servers due to invalid characters	1136
Limitations of server target validation	1137
Limitation of WebSphere Application Servers when workspace path begins with a backslash	1138
Limitation on the workbench to work with a WebSphere Application Server that changed its host name	1139
Limitations of servers due to long directory names	1140
The server might not stop completely when stopping the server from the Servers view	1141
A timeout error displays when starting the server immediately after a stop request is issued	1142
Limitations of X on Linux when you are using the WebSphere Application Server	1143
A runtime problem occurs when a Web 2.4 project requests EJB 3.0 injections	1144
Universal Test Client: Internal browser does not support IPv6 address	1145
Workspace migration limitation	1146
Web project wizard limitation	1147
Port connections or secured server	1148

Server port in use when you are restarting the development environment	1148
Limitations of desktop firewalls	1149
Long delays when establishing RMI connection after losing network connectivity	1150
The internal browser cannot display the administrative console for a secured WebSphere Application Server	1151
Administrative console login fails when running in secure mode	1152
Problems working with a secured server using SSL connections	1153
Unable to connect to a remote WebSphere Application Server hosted on a Red Hat Enterprise Linux machine	1153
Server connection or publishing problems when connecting to a remote WebSphere Application Server	1155
Unable to bind to ports on a Linux operating system after restarting the server	1156
Adding, removing or publishing applications to the server	1157
Troubleshooting if a problem is related to the workbench or the application server	1157
Limitation on Java 2 security application policy on WebSphere Application Servers	1157
Removal of an EJB module shared in multiple EAR projects	1158
Changing a Web or EJB project results in an application to be restarted with option Never publish automatically	1159
Limitations of Microsoft SQL Server JDBC driver when publishing to a remote WebSphere Application Server	1160
Publishing fails with duplicate class error	1161
Unable to use the workbench to publish the same application that was installed by using the administrative console	1162
Invalid profile name	1163
Universal Test Client: Internal browser does not support IPv6 address	1164
Using the Administrative Console	1164
The internal browser cannot display the administrative console for a secured WebSphere Application Server	1164
Administrative console login fails when running in secure mode	1164
Universal Test Client: Internal browser does not support IPv6 address	1164
Unable to use the workbench to publish the same application that was installed by using the administrative console	1164
Must restart server after changing the WebSphere Application Server configuration	1164

IBM WebSphere Developer Tools, Version 8.5.5

View the latest WebSphere® Developer Tools documentation. This information applies to Version 8.5.5 and to all subsequent releases and modifications until otherwise indicated in new editions.

Getting started

[Installing WebSphere Application Server Developer Tools for Eclipse](#)

[Tutorials](#)

[Samples](#)

[System requirements for WebSphere Application Server Developer Tools for Eclipse](#)

[Library page \(documentation in PDF format and more\)](#)

[Roadmap for WebSphere Application Server](#)

Common tasks

[Developing enterprise applications](#)

[Developing OSGi applications](#)

[Developing Apache Maven projects](#)

[Developing web applications](#)

[Developing web service applications](#)

[Testing and publishing on a server](#)

Troubleshooting and support

[Limitations and troubleshooting tips for Server tools](#)

[IBM Support Assistant](#)

[Support Technotes](#)

[WebSphere Application Server support](#)

[IBM Software Support home page](#)

[All groups on developerWorks](#)

More information

[↗ IBM Redbooks - WebSphere](#)

[↗ WebSphere on developerWorks](#)

[↗ Global WebSphere Community](#)

[↗ WASdev](#)

[↗ IBM Twitter, Facebook, and Blog information for WebSphere and CICS Support](#)

© Copyright IBM Corporation 2014

Overview: IBM WebSphere Application Server Developer Tools for Eclipse, Version 8.5.5

IBM® WebSphere® Application Server Developer Tools for Eclipse is a lightweight set of tools for developing, assembling, and deploying Java™ EE, OSGi, Web 2.0, and mobile applications to WebSphere Application Server.

When combined with WebSphere Application Server V8.5 Liberty Profile, WebSphere Application Server Developer Tools for Eclipse Version 8.5.5 is designed to provide a fast and lightweight environment for the rapid development and testing of Java EE web profile, Web 2.0, mobile, and OSGi applications.

For more information about IBM WebSphere Application Server Developer Tools for Eclipse, see:

- [8.5.5.6 What's new in Version 8.5.5.6](#)
- [8.5.5.5 What's new in Version 8.5.5.5](#)
- [8.5.5.4 What's new in Version 8.5.5.4](#)
- [8.5.5.2 What's new in Version 8.5.5.2](#)
- [What's new in Version 8.5.5](#)
- [Supported programming models](#)
- [Supported server tasks](#)
- [Getting started](#)
- [Getting help](#)
- [Additional features available with Rational Application Developer for WebSphere Software](#)
- [Using the Liberty profile as an application development environment](#)

[8.5.5.6](#) **What's new in Version 8.5.5.6**

Tools are available in this release to support:

- Enterprise JavaBeans (EJB) 3.2
- Generic Service Client (GSC) 8.7
- Bean Validation
- Contexts and Dependency Injection (CDI) 1.2
- Java EE Batch Tools
- Java EE permissions editor

[8.5.5.5](#) **What's new in Version 8.5.5.5**

Tools are available in this release to support:

- WebSocket 1.1

[8.5.5.4](#) **What's new in Version 8.5.5.4**

Tools are available in this release to support:

- WebSocket 1.0
- Servlet 3.1, including packaging as OSGi WABs
- Liberty Remote Server

[8.5.5.2](#) **What's new in Version 8.5.5.2**

Tools are available in this release to support:

- Dojo Version 1.9.2 adoption
- IBM SDK Version 7.1 for WebSphere Application Server

For information about features that were deprecated, removed, stabilized, and superseded in this release, see [What has changed in this release](#) .

What's new in Version 8.5.5

Tools are available in this release to support:

- Developing JAX-WS applications for the WebSphere Application Server Version 8.5.5 Liberty Profile
- Integrating OSGi bundles with Maven
- Improved validation of JCDI managed beans
- Simplified installation by dragging and dropping WebSphere Application Server Version 8.5.5 Liberty Profile
- Simplified download and installation of extensions for a WebSphere Application Server Liberty Profile
- The **minify** include option for packaging and sharing a WebSphere Application Server Liberty Profile runtime environment and configuration
- Improved support for developing User Features for extending the Liberty profile server functions
- Developing CDI applications
- Developing EJB 3.0 and 3.1 applications for the WebSphere Application Server Version 8.5.5 Liberty Profile
- Developing User Features for extending the Liberty profile server functions
- Integrating with Apache Maven

For information about features that were deprecated, removed, stabilized, and superseded in this release, see [What has changed in this release](#) .

Supported programming models

Programming model	WebSphere Application Server V8.5 Liberty Profile	WebSphere Application Server V8.5	WebSphere Application Server V8.0	WebSphere Application Server V7.0
Developing Java EE applications	Yes* Attention : *The tools for developing EJB applications support EJB 3.0 and 3.1.	Yes	Yes	Yes
Develop CDI applications	Yes*	Yes	Yes	No
Develop JAX-RS applications	Yes	Yes	Yes	Yes
Developing Web service applications	Yes*	Yes	Yes	Yes
Develop Web 2.0 and mobile web applications	Yes	Yes	Yes	Yes

Developing OSGi applications	Yes Note: The tools for developing OSGi applications support the following technologies: Web 2.5Web 3.0 8.5.5.4 Web 3.1JPAJAX-RSJSF	Yes Note: The tools for developing OSGi applications support the following technologies: Web 2.5Web 3.0JPAJAX-RSJSFEJB	Yes Note: The tools for developing OSGi applications support the following technologies: Web 2.5Web 3.0JPAJAX-RSJSF	No
------------------------------	--	--	---	----

The star icon (*) indicates that support is available starting in Version 8.5.5 release for WebSphere Application Server Liberty Profile.

Supported server tasks

The following features are supported on all WebSphere Application Servers:

- Migrate the server when you import applications that are targeted to a server that is not valid (except WebSphere Application Server Liberty Profile).
- Start and stop a remote server (including Version 8.5.5.4 release for WebSphere Application Server Liberty Profile and later).
- Run and debug administrative script files on the server (except WebSphere Application Server Liberty Profile).

Note: The workbench does not support WebSphere Application Server V7.0 feature packs and managed (federated) WebSphere Application Server Network Deployment environments.

Getting started

To get started, see [Installing WebSphere Application Server Developer Tools for Eclipse](#). See the [WASdev community](#) site:

- To participate in or post questions in the forum.
- To learn more about the workbench from the collection of articles and sample code.
- To get community news through blog entries.
- To help improve the workbench by reporting defects and requesting feature enhancements.

Getting help

WebSphere Application Server Developer Tools for Eclipse provides you with access to the Rational® Application Developer documentation. **Note:** Some documented features are available only with the full Rational Application Developer for WebSphere Software product.

For details about how to integrate the workbench with the server, see [Testing and publishing on a server](#).

Additional features available with Rational Application Developer for WebSphere Software

For enterprise-level development, consider using Rational Application Developer as your integrated development environment. Rational Application Developer provides a complete environment for Java, Java EE, web, web services, SOA, OSGi, Web 2.0, mobile, and portal designers and developers, including the server development tools. It also provides integration and support for the following application servers and associated feature packs:

- WebSphere Application Server V8.5 Liberty Profile
- WebSphere Application Server V8.5
- WebSphere Application Server V8.0
- WebSphere Application Server V7.0

Rational Application Developer is designed to accelerate development and unit test to ensure delivery of higher quality applications. It offers tools for teams that are using the emerging trends of Web 2.0, SOA, OSGi, and Cloud computing that help accelerate adoption and software delivery.

For more information about this product, see [Rational Application Developer for WebSphere Software](#).

- **Developer Tools known restrictions**

Several known restrictions apply when you are working with the WebSphere Developer Tools.

- **Developer Tools deprecations**

These features are deprecated in WebSphere Developer Tools as of the indicated version.

- **Notices**

- **Privacy Policy Considerations**

Developer Tools known restrictions

Several known restrictions apply when you are working with the WebSphere Developer Tools.

List of known issues and restrictions:

- Projects that are required by web fragment projects are not automatically added to the Java build path
- **8.5.5.4** Unable to import WAR file with no web.xml deployment descriptor and with Liberty profile server configured with JRE version 6 or earlier as target runtime
- **8.5.5.4** Unable to import EJB jar file with no ejb-jar.xml deployment descriptor and with Liberty profile server configured with JRE version 6 or earlier as target runtime
- **8.5.5.4** Remote Liberty profile server remains in Publishing state with the message Waiting for application status from the server
- **8.5.5.6** In a Liberty profile server, changes to method annotations are not automatically updated in debug mode

Projects that are required by web fragment projects are not automatically added to the Java build path

If you have a web application project that contains one or more web fragment projects, and the web application project has references in its deployment assembly page to projects that are required by the web fragment projects, then those dependent projects (such as the JPA and Utility projects) are not automatically added to the Java™ build path of the web fragment projects.

Manually add the dependent projects to the Java build path of the web fragment projects in order for them to compile.

8.5.5.4 Unable to import WAR file with no web.xml deployment descriptor and with Liberty profile server configured with JRE version 6 or earlier as target runtime

If you attempt to import a WAR file using the option **File > Import > Web > WAR file**, and the WAR file you select does not have a deployment descriptor, `web.xml`, and you select a Liberty profile server configured with a JRE version 6 or earlier as the target runtime, the **OK** button is not enabled, and no error is displayed in the wizard.

8.5.5.4 Unable to import EJB jar file with no ejb-jar.xml deployment descriptor and with Liberty profile server configured with JRE version 6 or earlier as target runtime

If you attempt to import a EJB jar file using the option **File > Import > EJB > EJB JAR file**, and the EJB jar file you select does not have a deployment descriptor, `ejb-jar.xml`, and you select a Liberty profile server configured with a JRE version 6 or earlier as the target runtime, the **OK** button is not enabled, and no error is displayed in the wizard.

8.5.5.4 Remote Liberty profile server remains in Publishing state with the message waiting for application status from the server

When doing a **Publish** to push updates to an application, the operation stalls with a `waiting for application status from the server` message. This is a situation where the developer tools miss notifications from the server. The **Publish** may be complete but because the tools do not receive the notification, the operation continues to wait.

In the Console view, you will see a log message for the application you published. When the update has completed, you should see an `application updated` message for this application. This indicates that the server has completed the work. You can cancel the progress monitor in the **Progress** view and proceed normally.

8.5.5.6 In a Liberty profile server, changes to method annotations are not automatically updated in debug mode

See [The hot method replace of debugging for a WebSphere Application Server](#) for more information about this limitation.

Parent topic: [Overview: IBM WebSphere Application Server Developer Tools for Eclipse, Version 8.5.5](#)

Notices

This information was developed for products and services that are offered in the U.S.A.

IBM may not offer the products, services, or features that are discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM® product, program, or service.

IBM may have patents or pending patent applications covering subject matter that is described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to: Intellectual Property Licensing

Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact: IBM Corporation

Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us. Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems.

Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

APACHE INFORMATION. The information center includes all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions that are made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org>. You may obtain a copy of the Apache License at

<http://www.apache.org/licenses/LICENSE-2.0>.

Programming interface information

This publication primarily documents information that is NOT intended to be used as Programming Interfaces of WebSphere® Application Server. This publication also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of WebSphere Application Server. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking: Programming Interface information.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "[Copyright and trademark information](#)" at www.ibm.com/legal/copytrade.shtml.

Parent topic: [Overview: IBM WebSphere Application Server Developer Tools for Eclipse, Version 8.5.5](#)

Privacy Policy Considerations

IBM® Software products, including software as a service solutions, (Software Offerings) can use cookies or other technologies to collect product usage information, to help improve the user experience, to tailor interactions with the user or for other purposes. In many cases, no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies follows.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from users using cookies and other technologies, seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's privacy policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details/us/en> in the Cookies, Web Beacons and Other Technologies and Software Products and Software-as-a Service sections.

Parent topic: [Overview: IBM WebSphere Application Server Developer Tools for Eclipse, Version 8.5.5](#)

Tutorials

A tutorial provides step by step instructions for you to follow, to complete specific tasks.

The tutorials for IBM® WebSphere® Application Server Developer Tools are contained in the following subsections:

- **Java tutorials**

The Java™ tutorials provide step by step instructions for you to follow, to complete specific tasks relating to Java applications.

- **OSGi tutorials**

The OSGi tutorials provide step by step instructions for you to complete specific tasks that relate to OSGi applications.

- **Web tutorials**

The Web tutorials provide step by step instructions for you to follow, to complete specific tasks relating to Web applications.

- **Web services tutorials**

The Web services tutorials provide step by step instructions for you to follow, to complete specific tasks relating to Web service applications.

Java tutorials

The Java™ tutorials provide step by step instructions for you to follow, to complete specific tasks relating to Java applications.

The Java tutorials are contained in the following subsections:

- **[Tutorial: Create a Hello World Java application](#)**

This tutorial shows you how to create a simple Java application using the WebSphere® Developer Tools product.

Parent topic: [Tutorials](#)

[Tutorials](#) > [Java tutorials](#) >

[< Previous](#) | [Next >](#)

Tutorial: Create a Hello World Java application

This tutorial shows you how to create a simple Java™ application using the WebSphere® Developer Tools product.

Learning objectives


View the video for this tutorial to learn how to:

- Create a Java project.
- Create a Java package.
- Create a Java class.
- Run a Java application.

Time required

10 minutes

Video

 **Watch:** [Creating a Hello World Java application](#) shows how to create and test a Java application using WebSphere Developer Tools V8.5.5.2 and later. [\[Transcript\]](#)

[< Previous](#) | [Next >](#)

OSGi tutorials

The OSGi tutorials provide step by step instructions for you to complete specific tasks that relate to OSGi applications.

The OSGi tutorials are contained in the following subsections:

- [OSGi EJB tutorial](#)

This tutorial shows you how to create an OSGi application that exposes an EJB as a service. You learn how to create OSGi bundles with EJB support, use the OSGi tools to manage EJB exports, and create a servlet that accesses the EJB as an OSGi service. You create the same application that is demonstrated in the OSGi EJB temperature converter sample.

- [Develop a simple OSGi application](#)

This tutorial demonstrates how to create an OSGi application and run it on either WebSphere® Application Server full profile or Liberty profile. The OSGi application consists of an OSGi web bundle that contains a servlet that accesses a service that is provided in another bundle project. This tutorial is an introduction to using OSGi application development tools.

Parent topic: [Tutorials](#)

OSGi EJB tutorial

This tutorial shows you how to create an OSGi application that exposes an EJB as a service. You learn how to create OSGi bundles with EJB support, use the OSGi tools to manage EJB exports, and create a servlet that accesses the EJB as an OSGi service. You create the same application that is demonstrated in the OSGi EJB temperature converter sample.

Learning objectives

Important: Applicable edition: Full profile

In this tutorial, you learn how to create an OSGi application that exposes an EJB as a service. It demonstrates how to create OSGi bundles with EJB support, use the OSGi tools to manage EJB exports, and create a servlet that accesses the EJB as an OSGi service. This tutorial shows how to create the same application that is demonstrated in the OSGi EJB temperature converter sample.

The application that is developed in this tutorial is a simple temperature conversion application that requires four projects:

- An OSGi application, ConverterApp, to include the bundles that are developed.
- An OSGi bundle project, EJB, that has EJB support. The EJB is exposed as a service by using the OSGi header Export-EJB.
- An EJB client project, EJBClient, to contain the interface code for the EJB. EJBClient has OSGi bundle support.
- An OSGi bundle project, Web, that has Web 3.0 support. This project includes a servlet that is configured to access the EJB that is exposed as an OSGi service.

To complete this tutorial, you need approximately 60 minutes.

When you are ready, begin [Lesson 1: Create an OSGi project with EJB support](#)

Related information:

[OSGi EJB temperature converter sample](#)

Lesson 1: Create an OSGi project with EJB support

In this lesson, you create the initial projects that are required for the application. You use the OSGi bundle project wizard to create: a bundle with EJB support, an EJB client project to contain client interfaces, and an OSGi application that includes the bundles.

1. Launch the OSGi bundle project wizard. Click **File > New > OSGi Bundle Project**.
2. Create a project with EJB support. Enter the following values in the OSGi bundle project wizard:
 - **Project name**
 - EJB
 - **Target runtime**
 - WebSphere® Application Server v8.5.
 - **Add EJB support**
 - Check **Add EJB support** and for this tutorial select **EJB 3.1**.
 - **Add bundle to application > Application project**
 - Check **Add bundle to application** and for **Application project** enter `ConverterApp`.

OSGi Bundle Project

Create a standalone OSGi bundle project or add it to a new or existing application.

Project name:

Project location

Use default location

Location:

Target runtime

Configuration

Add Web support

Add persistence support

Add EJB support

Custom

Generate blueprint file

Application membership

Add bundle to application

Application project:

Working sets

Add project to working sets

- Click **Next** until you are at the EJB Module screen. Ensure that the box **Create an EJB Client JAR module to hold the client interfaces and classes** is checked. Ensure that **Name** is `EJBClient` and **Client JAR URI** is `EJBClient.jar`.

EJB Module

Configure EJB module settings.

EJB Client JAR

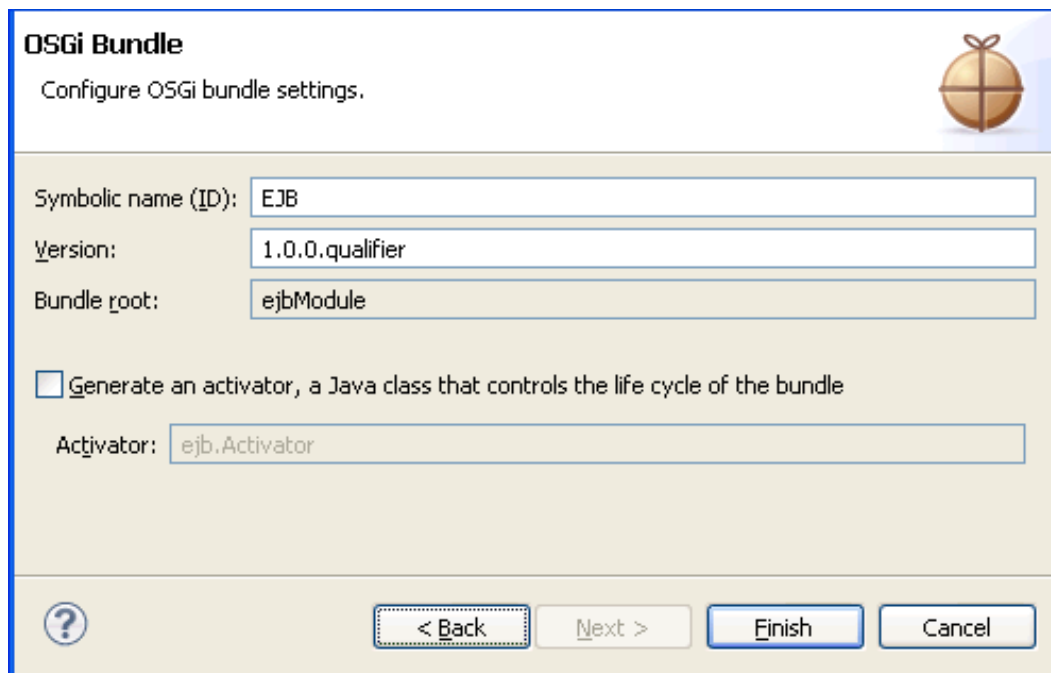
Create an EJB Client JAR module to hold the client interfaces and classes

Name:

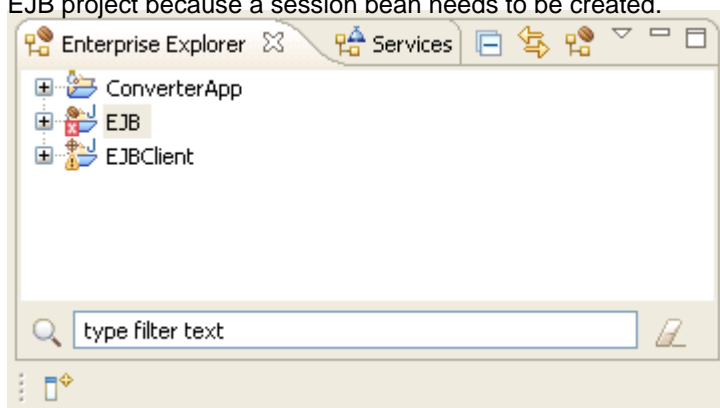
Client JAR URI:

Generate ejb-jar.xml deployment descriptor

- Click **Next** to go to the OSGi Bundle screen. Accept the default values.



5. Click **Finish**. The ConverterApp, EJB, and EJBClient projects are created. At this point, there might be an error in the EJB project because a session bean needs to be created.



[< Previous](#) | [Next >](#)

Lesson 2: Develop an EJB

In this lesson you create a session bean, develop a local interface, and develop the implementation class for the bean. You also use the manage EJB exports dialog to expose the EJB as an OSGi service.

Creating a session bean

In this section, you create a session bean and specify the local interface.

1. Open the session bean wizard. In the EJB project, right-click `ejbModule` and select **New > Session Bean (EJB 3.x)**.

2. Create the bean. Enter the following values in the wizard:

- Project

- EJB

- Java™ package

- `com.ibm.example.impl`

- Class name

- `EJBConverter`

- State type

- Stateless

- Local

- Select the **Local** check box and enter the following value: `com.ibm.example.EJBConverterLocal`

- No-interface View

- Clear **No-interface View**

Create EJB 3.x Session Bean
Specify class file destination.

Project: EJB

Source folder: /EJB/ejbModule

Java package: com.ibm.example.impl

Class name: EJBConverter

Superclass:

State type: Stateless

Create business interface

Remote com.ibm.example.impl.view.EJBConverterRemote

Local com.ibm.example.EJBConverterLocal

No-interface View

Note: The application that is developed in this tutorial uses a local interface. If you want to use remote interfaces, select the **Remote** check box in the wizard and then develop the corresponding interface. JNDI lookups for remote interfaces work differently. With remote interfaces, adjust the lookup code. The next lesson will give an example.

3. Click **Finish**. The session bean is created and the class opens in the editor.

Developing the local interface

When you created the session bean, the `EJBConverterLocal` interface was created in the `EJBClient` project. In this section, you add two methods to the interface. The methods describe basic temperature conversion operations.

1. Open the `EJBConverterLocal` interface. In the `EJBClient` project, expand the `ejbModule` folder and the `com.ibm.example` package. Double-click the `EJBConverterLocal` interface. The interface opens in the editor.

2. Add methods to the interface. Add the following methods to the interface:

```
public double convertToFahrenheit(double celsius);
```

```
public double convertToCelsius(double fahrenheit);
```

The following code is the result:

```
package com.ibm.example;
```

```
public interface EJBConverterLocal {

    public double convertToFahrenheit(double celsius);

    public double convertToCelsius(double fahrenheit);

}
```

3. Save your changes. Note that when you save the interface file, errors are displayed for the EJBConverter class because it does not yet implement the methods that you added to the interface.

Developing the implementation class

In this section, you implement the temperature conversion methods that are required by the local interface.

1. Open the EJBConverter class. In the EJB project, expand the ejbModule folder and the com.ibm.example.impl package. Double-click the EJBConverter class. The class opens in the editor.
2. Remove the default constructor.
3. Add the implementations for the two methods you added to the EJBConverterRemote interface. Add the following

```
method implementations to the class:public double convertToCelsius(double fahrenheit) {  
    return (fahrenheit - 32) * (5 / 9d);  
}  
  
public double convertToFahrenheit(double celsius) {  
    return celsius * (9 / 5d) + 32;  
}
```

The completed class looks like the following code. Note that the original code generated by the wizard contains an annotation, `@Stateless`, and an annotation, `@Local`, that specifies the local interface that you entered in the session bean wizard.`package com.ibm.example.impl;`

```
import com.ibm.example.EJBConverterLocal;  
import javax.ejb.Local;  
import javax.ejb.Stateless;  
  
@Stateless  
@Local(EJBConverterLocal.class)  
public class EJBConverter implements EJBConverterLocal {  
  
    public double convertToCelsius(double fahrenheit) {  
        return (fahrenheit - 32) * (5 / 9d);  
    }  
  
    public double convertToFahrenheit(double celsius) {  
        return celsius * (9 / 5d) + 32;  
    }  
}
```

4. Save your changes.

Exposing an EJB as an OSGi service

By default, EJBs that you add to an OSGi project that has EJB support are automatically exposed as OSGi services.

Exposing OSGi services is controlled through the Export-EJB header in the OSGi manifest. In this section, you check the manifest and use the manage EJB exports dialog to confirm that EJBConverter is exposed as an OSGi service.

1. Open the manifest. In the EJB project, double-click **Manifest:EJB**. The manifest opens. Click the **MANIFEST.MF** tab to view the file in text format. Note that there is an **Export-EJB** header with an entry for the EJBConverter EJB. Note that you can manage EJB exports by manually adding entries to the Export-EJB header. EJBs can be added as a comma-separated list. **Note:** There are two special cases of the Export-EJB header to be aware of:

- NONE

- If you specify `NONE` as an entry for the Export-EJB header, no EJBs are exposed as services. If you specify `NONE`, but then also add an EJB to the list, a warning is displayed by the tools.

- BLANK

- If you have an Export-EJB header in the manifest but there are no entries, by default all EJBs in the project are exposed as services.

2. Open the manage EJB export dialog. Right-click the EJB project and select **OSGi > Expose EJBs as OSGi services**. The manage EJB exports dialog opens. In the dialog, confirm that the box next to the EJBConverter EJB is selected and click **OK**. You can use this dialog in your projects to add and remove EJBs that are exposed as services. Changing the EJBs selected in this dialog changes the entries to the Export-EJB header in the manifest.



Note: If you create new EJBs to the project, they are automatically exposed as OSGi services. You can use the manage EJB exports dialog to clear any EJBs that you do not want to expose.

[< Previous](#) | [Next >](#)

Lesson 3: Develop a web bundle that accesses the EJB

In the previous lesson, you exposed the EJB as an OSGi service. In this lesson, you develop a web bundle that accesses the service. You create the bundle, configure the manifest to import required packages, develop a servlet that accesses the EJB as an OSGi service, and create an HTML page to access the application through a browser.

Creating a web bundle project

In this section, you create an OSGi bundle with Web 3.0 support and add the bundle to the ConverterApp OSGi application.

1. Open the OSGi bundle project wizard. Click **File > New > OSGi Bundle Project**. The wizard opens.
2. Create the project. Enter or select the following values in the wizard:

- **Project name**

 - Web

- **Target run time**

 - WebSphere Application Server v8.5.

- **Add Web support (select this check box and select the following value)**


 - Web 3.0.

- **Add bundle to application (select this check box and select the following application)**

 - ConverterApp

OSGi Bundle Project

Create a standalone OSGi bundle project or add it to a new or existing application.



Project name:

Project location

Use default location

Location:

Target runtime

Configuration

Add Web support

Add persistence support

Add EJB support

Custom

Generate blueprint file

Application membership

Add bundle to application

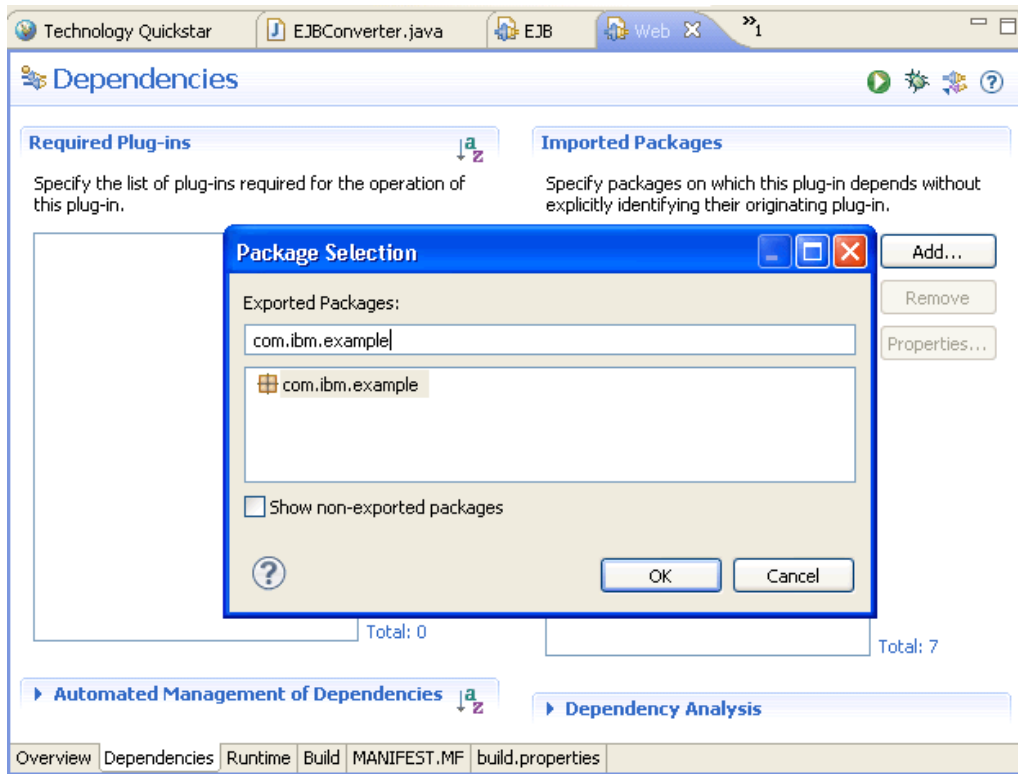
Application project:

3. Click **Finish**. The project is created.

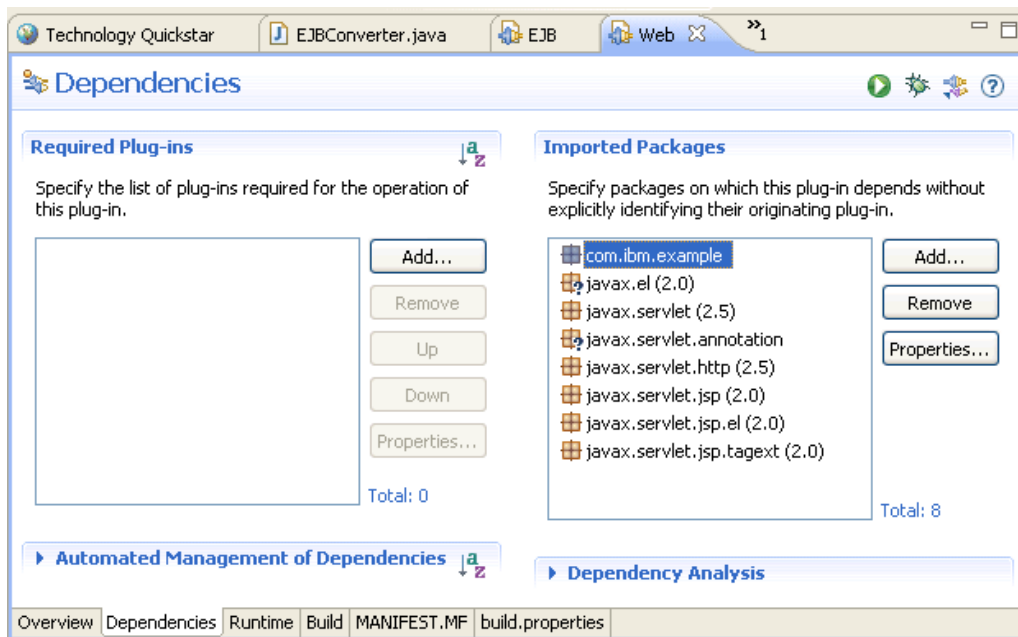
Configuring the bundle manifest in the web project to import the package from the EJBClient project

In this section, you add an entry for the `com.ibm.example` package in the EJBClient project to the Import-Package header in the manifest. The `com.ibm.example` package contains the interface that you developed for the EJB.

1. Open the manifest. In the web project, double-click **Manifest: Web**. The manifest editor opens.
2. Add a dependency to the `com.ibm.example` package. Click the **Dependencies** tab. In the **Imported Packages** section, click **Add**. The package selection dialog opens. In the **Exported Packages** field, enter `com.ibm.example`. Select `com.ibm.example` from the package list and click **OK**.



The dependency to `com.ibm.example` is added to the Imported Packages section. Note that because the bundle project is configured with Web 3.0 support, extra import entries for servlet packages such as `javax.servlet` are already added.



3. Save your changes.

Creating a servlet

In this section, you create an initial servlet with the servlet wizard.

1. Open the servlet wizard. Right-click the web project and select **New > Servlet**. The servlet wizard opens.
2. Create the servlet. Enter or select the following values in the wizard:

- **Project**

- Web

- Java™ package

- com.ibm.example.servlets

- Class name

- ConverterServlet

Click **Finish**. The ConverterServlet is created in the web project.

Developing the servlet

In this section, you develop the servlet and include code to access the EJB as an OSGi service.

1. Open the servlet in the Java editor. If the servlet is not already open in the editor, in the web project, expand the Java Resources/src folder. In the com.ibm.example.servlets package, double-click the ConverterServlet.
2. Add new import statements. Add the following import statements to the import section at the beginning of the source file:

```
import com.ibm.example.EJBConverterLocal;
import java.io.PrintWriter;
import java.text.NumberFormat;
import javax.naming.InitialContext;
import javax.naming.NamingException;
```

3. Develop the doGet() method. The servlet contains a basic doGet() method. Replace the auto-generated method with the following method:

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {

    PrintWriter w = response.getWriter();

    NumberFormat nf = NumberFormat.getInstance();
    nf.setMaximumFractionDigits(2);

    try {
        InitialContext context = new InitialContext();
        EJBConverterLocal converter = (EJBConverterLocal) context
            .lookup("osgi:service/" + EJBConverterLocal.class.getName());

        String temperature = request.getParameter("temperature");
        if (temperature == null) {
            w.println("<p>A temperature value was not specified.</p>");
        } else if (!temperature.matches("[+]?[0-9]*\\.?[0-9]+")) {
            w.println("Invalid temperature specified.");
        } else {
            double degrees = Double.parseDouble(temperature);

            double celsius = converter.convertToCelsius(degrees);
            w.println("<p>" + temperature + " degrees fahrenheit is "
                + nf.format(celsius) + " degrees celsius</p>");

            double fahrenheit = converter.convertToFahrenheit(degrees);
            w.println("<p>" + temperature + " degrees celsius is "
                + nf.format(fahrenheit) + " degrees fahrenheit</p>");
        }
    }
```

```

        w.println("<a href='index.html'>Back</a>");
    } catch (NamingException e) {
        w.println(e.getMessage());
    } catch (NumberFormatException e) {
        w.println("An incorrect temperature was specified");
    }
}

```

Note: Items to note about this code: The servlet creates an `EJBConverterLocal` object called `converter`. Recall that `EJBConverterLocal` was the interface you created that contains two temperature conversion methods `convertToCelsius()` and `convertToFahrenheit()`. The following line of code handles creation of the object that accesses the EJB as an OSGi service: `EJBConverterLocal converter = (EJBConverterLocal) context.lookup("osgi:service/" +`

```
EJBConverterLocal.class.getName());
```

The temperature value to convert is passed to the servlet by the `temperature` parameter. This value is converted to a `Double` called `degrees`, which can be processed by the `convertToCelsius()` and `convertToFahrenheit()` methods on the `converter` object.

Note: If you want to use remote interfaces, adjust your JNDI lookup code. For example, the context lookup section previously described can be written in the following format: `EJBConverterRemote converter = (EJBConverterRemote)`

```
context.lookup("osgi:service/" + EJBConverterRemote.class.getName() + "/(service.exported.interfaces=*)");
```

4. Develop the `doPost()` method. Replace the auto-generated `doPost()` method with the following code. Replacing the auto-generated `doPost()` method ensures that the servlet can be called by both post and get requests from a browser. Calls to `doPost()` automatically execute the `doGet()` method.

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
}

```

5. Check your work. The following code is the completed servlet code: `package com.ibm.example.servlets;`

```

import java.io.IOException;
import java.io.PrintWriter;
import java.text.NumberFormat;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.example.EJBConverterLocal;

@WebServlet("/ConverterServlet")
public class ConverterServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ConverterServlet() {

```

```

    super();
}

protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {

    PrintWriter w = response.getWriter();

    NumberFormat nf = NumberFormat.getInstance();
    nf.setMaximumFractionDigits(2);

    try {
        InitialContext context = new InitialContext();
        EJBConverterLocal converter = (EJBConverterLocal) context
            .lookup("osgi:service/" + EJBConverterLocal.class.getName());

        String temperature = request.getParameter("temperature");
        if (temperature == null) {
            w.println("<p>A temperature value was not specified.</p>");
        } else if (!temperature.matches("[-+]?[0-9]*\\.?[0-9]+")) {
            w.println("Invalid temperature specified.");
        } else {
            double degrees = Double.parseDouble(temperature);

            double celsius = converter.convertToCelsius(degrees);
            w.println("<p>" + temperature + " degrees fahrenheit is "
                + nf.format(celsius) + " degrees celsius</p>");

            double fahrenheit = converter.convertToFahrenheit(degrees);
            w.println("<p>" + temperature + " degrees celsius is "
                + nf.format(fahrenheit) + " degrees fahrenheit</p>");
        }

        w.println("<a href='index.html'>Back</a>");
    } catch (NamingException e) {
        w.println(e.getMessage());
    } catch (NumberFormatException e) {
        w.println("An incorrect temperature was specified");
    }
}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
    doGet(request, response);
}
}

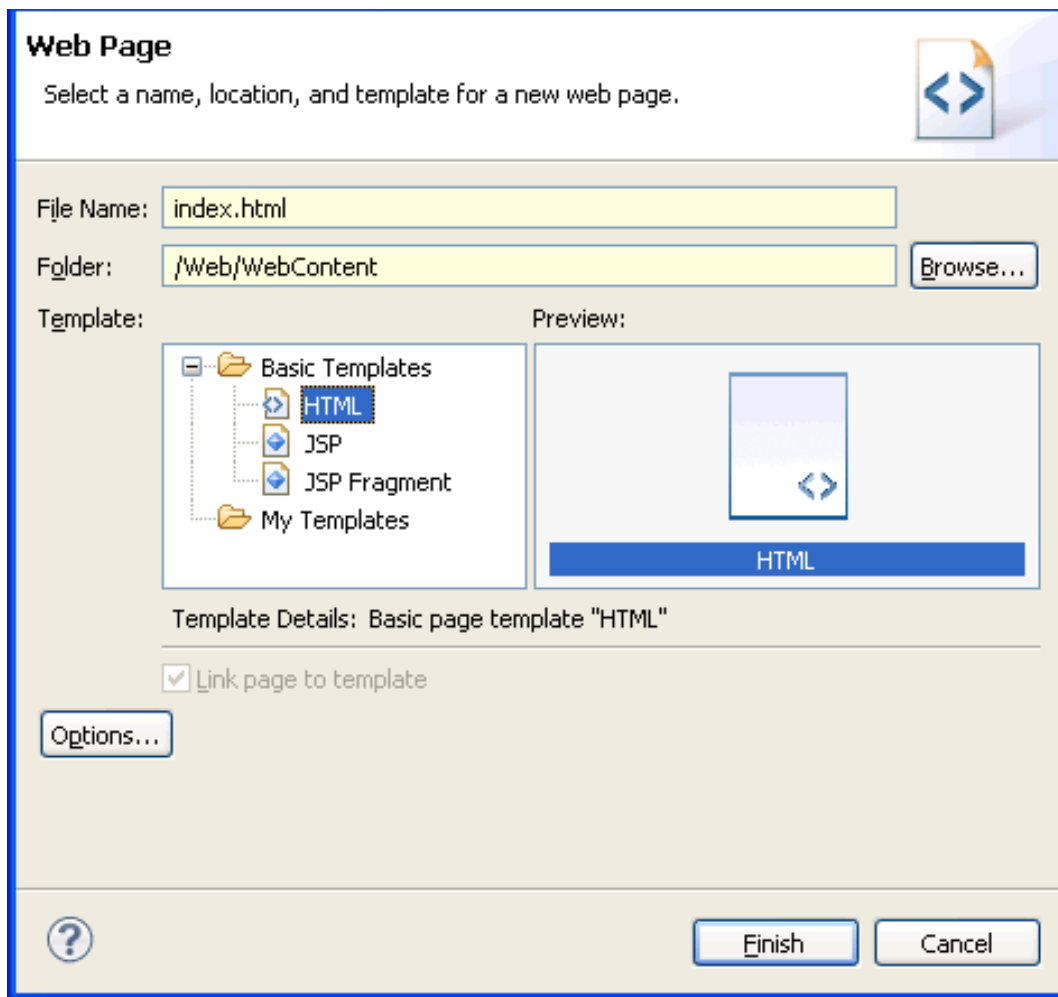
```

6. Save your changes.

Creating an HTML file to access the application

In this section, you create an HTML file, `index.html`, that contains a form for entering values to submit to the servlet for conversion.

1. Open the new Web page wizard. Right-click the web project and select **New > Web Page**. The Web page wizard opens.
2. Create the web page. In the wizard, in the **File Name** field, enter `index.html`. In the **Template** section, select **HTML**. Click **Finish**. The page is created and opens in the editor.



3. Select the **Source** tab of the web page editor.
4. Update the source. Replace the default source code with the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<title>OSGi EJB Temperature Converter</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">

function validate(form) {
    var temperature = new Number(form.temperature.value);
    if (isNaN(temperature)) {
        alert("Please enter a valid number.");
    }
}
```



```
    return false;
}
return true;
}

</script>
</head>
<body>
<form action="ConverterServlet" method="post" onsubmit="return validate(this);">
  Enter a temperature value:
  <br/>
  <input type="text" id="temperature" name="temperature"/>
  <br/>
  <br/>
  <input type="submit" value="Submit"/>
</form>
</body>
</html>
```

This HTML contains a form that submits a *temperature* parameter value to the ConverterServlet. Before it is sent to the servlet, the value to be submitted is validated by the `validate()` function to ensure that a numerical value was entered.

[< Previous](#) | [Next >](#)

[< Previous](#)

Lesson 4: Test the application

In this lesson, you test the application that you developed in the preceding lessons. You ensure that you have an application server instance that is set up to run the application, run the application on the server, and submit test values for temperature conversion.

Checking servers view for the application server

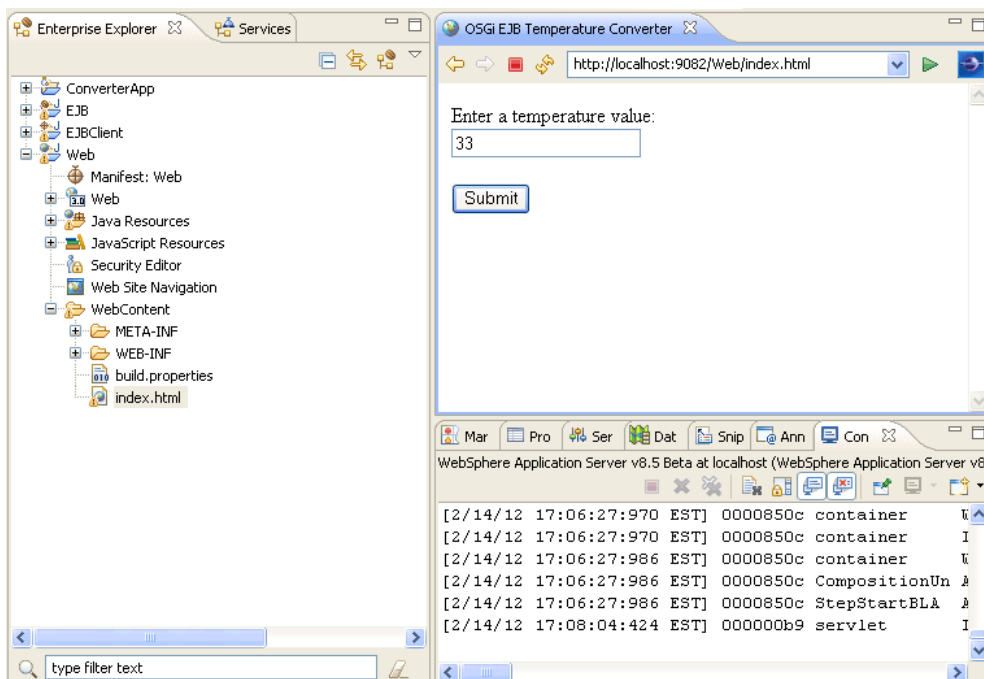
In this section, you ensure that you have a WebSphere Application Server v8.5 instance that is set up to run the OSGi application.

1. Click the **Servers** tab to access the servers view.
2. Check that you have a WebSphere Application Server v8.5 server. If you do not have a server, right-click in the **Servers** view and select **New > Server**. Choose **WebSphere Application Server v8.5** from the list and follow the rest of the steps of the wizard.
3. Check that the server is running. If the server is running, the status information next to the server says **Started**. If it is not running, to start it, right-click the server and select **Start**.

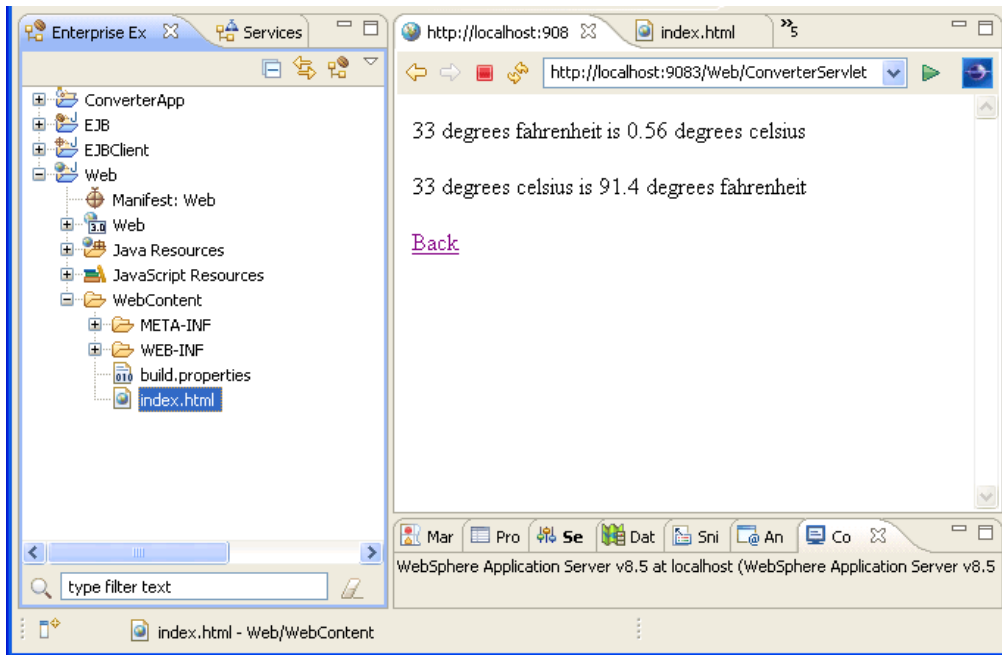
Running the application on the server

In this section, you run the application and submit test values.

1. Launch the application through the `index.html` file. In the web project, right-click the `index.html` file that you created and select **Run As > Run on Server**. Choose the WebSphere Application Server v8.5 server from the server list and click **Finish**. The application is deployed to the server and the deployed `index.html` file opens in the embedded browser.
2. In the field on the page, enter a temperature value that you want to convert.



3. Click **Submit**. The value is converted. The processing is handled by the `EJBConverter` EJB that is available as an OSGi service.



[< Previous](#)

Develop a simple OSGi application

This tutorial demonstrates how to create an OSGi application and run it on either WebSphere® Application Server full profile or Liberty profile. The OSGi application consists of an OSGi web bundle that contains a servlet that accesses a service that is provided in another bundle project. This tutorial is an introduction to using OSGi application development tools.

Learning objectives

In this tutorial, the following learning objectives are met:

- Create an OSGi bundle and an OSGi application.
- Create and configure an OSGi blueprint configuration file.
- Configure component dependencies.
- Add dependency injection within a bundle.
- Add dependency injection between bundles.
- Deploy an OSGi application to a server.

Time required

This tutorial takes approximately 90 minutes to finish. If you explore other concepts that are related to this tutorial, it might take longer to complete.

Related information:

[Sample: OSGi Counter Service](#)

Introduction: Develop a simple OSGi application

This tutorial demonstrates how to create an OSGi application and run it on either WebSphere® Application Server full profile or Liberty profile. The OSGi application consists of an OSGi web bundle that contains a servlet that accesses a service that is provided in another bundle project. This tutorial is an introduction to using OSGi application development tools.

Learning objectives

In this tutorial, the following learning objectives are met:

- Create an OSGi bundle and an OSGi application.
- Create and configure an OSGi blueprint configuration file.
- Configure component dependencies.
- Add dependency injection within a bundle.
- Add dependency injection between bundles.
- Deploy an OSGi application to a server.

Time required

This tutorial takes approximately 90 minutes to finish. If you explore other concepts that are related to this tutorial, it might take longer to complete.

Prerequisites

Install WebSphere Application Server. **Tip:** You can run this tutorial on WebSphere Application Server Versions 7.0, 8.0, 8.5 and 8.5 Liberty Profile.

Learn more about installing WebSphere Application Server Version 7.0: To run this sample on WebSphere Application Server Version 7.0, you must install the Feature Pack for OSGi Applications and Java™ Persistence API 2.0.

To Install the feature pack:

1. Open the IBM® Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 7.0 Test Environment**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 7.0.
6. In the Features list, ensure that you select **OSGi Applications** under **IBM WebSphere Application Server Version 7.0 Feature Pack for OSGi Applications and Java Persistence API 2.0**.

Lessons in this tutorial

- [Lesson 1: Create the bundle and application](#)

An OSGi bundle is a Java archive file that contains Java code, resources, and a manifest that describes the bundle and its dependencies. An OSGi bundle contains the business logic and metadata that you need to run a service. A bundle is a module in an application, which in turn is deployed to a server.

- [Lesson 2: Develop the business logic](#)

- [Lesson 3: Create the blueprint configuration file](#)

The blueprint configuration file contains the component assembly and configuration information for a bundle. The file describes how components are registered in the OSGi service registry or how they look up services from the OSGi

service registry. This information is used at run time to instantiate and configure the required components when the bundle is started. In this tutorial, the blueprint file defines a service that other components can use to access the counter that is defined in Lesson 2.

- **Lesson 4: Create a servlet that accesses an OSGi service**

- **Lesson 5: Deploy the OSGi Counter application**

Follow these steps to deploy the application to a WebSphere Application Server installation that has the OSGi feature pack installed. Refer to the WebSphere Application Server documentation for installation and configuration details.

- **Lesson 6: Add dependency injection within a bundle**

Dependency injection allows one bean to access another bean without having to implement any code to create the bean instance. The required bean instance is created by the blueprint container by using information that is contained in the blueprint configuration file.

- **Lesson 7: Add dependency injection between bundles**

In the previous lesson, you learned how to use dependency injection within a bundle. Dependency injection between bundles requires an extra reference entry in the blueprint configuration file.

[< Previous](#) | [Next >](#)

< [Previous](#) | [Next](#) >

Lesson 1: Create the bundle and application

An OSGi bundle is a Java™ archive file that contains Java code, resources, and a manifest that describes the bundle and its dependencies. An OSGi bundle contains the business logic and metadata that you need to run a service. A bundle is a module in an application, which in turn is deployed to a server.

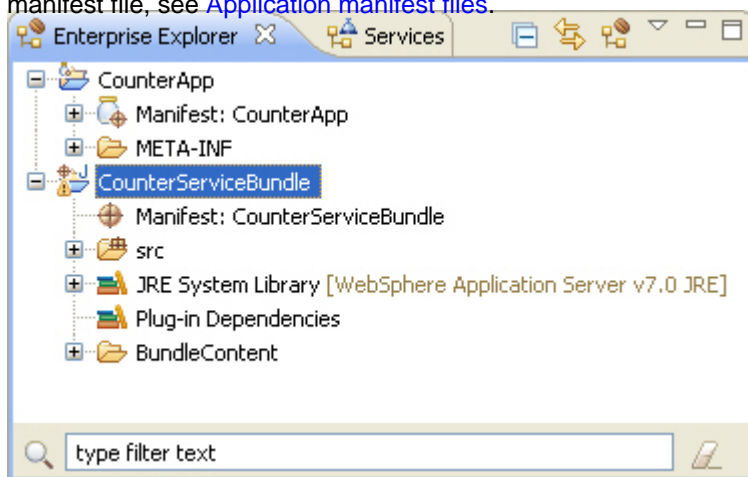
An OSGi application project groups a set of bundles to provide a coherent business logic. The application can consist of different bundle types such as web enabled bundles and persistence (JPA) enabled bundles.

In this lesson, you create an OSGi bundle that contains OSGi Blueprint information that defines a service that provides a plain old Java object (POJO) component assembly model. A POJO is an ordinary Java object, as distinguished from a special Java object, such as an enterprise entity bean.

To create the bundle, `CounterServiceBundle`:

1. Click **File** > **New** > **Other** and then expand **OSGi**.
2. Click **OSGi Bundle Project** and then click **Next**. The New OSGi Bundle Project opens.
3. In the **Project name** field, type `CounterServiceBundle`.
4. In the Target runtime list, select one of the following servers:
 - **WebSphere Application Server v7.0**
 - **WebSphere Application Server v8.0**
 - **WebSphere Application Server v8.5**
 - **WebSphere Application Server V8.5 Liberty Profile**
5. In the **Application project** field, change the name of your application project to `CounterApp` and then click **Finish**.

Your OSGi bundle project is created and a bundle manifest is added to your project. Your OSGi application project is also created and your application manifest is added to the project. The application manifest file contains metadata that enables the OSGi Framework to process the modular aspects of the bundles. For more information about the OSGi application manifest file, see [Application manifest files](#).



Learn more about the bundle manifest file:To view the bundle manifest, expand your project and then double-click **Manifest: CounterServiceBundle**. The bundle manifest opens in the editor. The bundle manifest source looks similar to the following example:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: CounterServiceBundle
Bundle-SymbolicName: CounterServiceBundle
Bundle-Version: 1.0.0
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

The OSGi bundle manifest file contains metadata that enables the OSGi Framework to process the modular aspects of the

bundle. For more information about the OSGi bundle manifest file, see [OSGi bundle manifest file](#).

Learn more about the application manifest: To view the application manifest, double-click **Manifest:CounterApp** to open your application manifest in the editor. The application manifest looks similar to the following example:

```
Name: CounterApp
Application-SymbolicName: CounterApp
Application-ManifestVersion: 1.0
Application-Version: 1.0.0
Manifest-Version: 1.0
Application-Content: CounterServiceBundle;version=1.0.0,
```

`Application-SymbolicName` is the OSGi application name. `Application-Content` lists the bundle names with the acceptable range of OSGi version specifications. In this tutorial, the bundle `CounterServiceBundle` is tolerated with a version of 1.0 or later.

Important: There must be a carriage return at the end of the last line of the `APPLICATION.MF` file.

For more information about the OSGi application manifest file, see [Application manifest files](#).

Lesson checkpoint

You created the `CounterServiceBundle` bundle and the `CounterApp` application.

In this lesson, you learned about the following topics:

- How to create an OSGi bundle project and OSGi application project.
- About the bundle manifest file.
- About the application manifest file.

[< Previous](#) | [Next >](#)

Lesson 2: Develop the business logic

In this lesson, you create the business logic for your simple OSGi application. The business logic uses a POJO component assembly model. This example application provides a simple counter that increments each time it is accessed.

To create the business logic:

1. [Creating the package.](#)
2. [Creating the interface class.](#)
3. [Creating the implementation class.](#)
4. [Exporting the package.](#)

Creating the package

1. In Enterprise Explorer, right-click **CounterServiceBundle/src** and then click **New > Package**. The New Java Package wizard opens.
2. In the **Name** field, type `com.ibm.ws.eba.counter` and then click **Finish**. The package is created in the `src` folder.

Creating the interface class

1. Right-click the package **com.ibm.ws.eba.counter** and then click **New > Interface**. The New Java Interface wizard opens.
2. In the **Name** field, type `Counter` and then click **Finish**. The interface is created in the package and it opens in the editor.
3. Add the `getCount()` method to the interface. The following code is the result:`package com.ibm.ws.eba.counter;`

```
public interface Counter {  
    public int getCount();  
}
```

4. Save `Counter.java`.

Creating the implementation class

1. Right-click the package **com.ibm.ws.eba.counter** and then click **New > Class**. The New Java Class wizard opens.
2. In the **Name** field, type `CounterImpl`.
3. Click **Add**. The Implemented Interfaces Selection dialog opens.
4. In the **Choose interfaces** field, type `Counter`. Select the **Counter** interface for `com.ibm.ws.eba.counter` and then click **OK**.
5. Click **Finish**. The class is created in the package and it opens in the editor.
6. Add the implementation for the `getCount()` method. Add an initialization method that confirms that the service starts on the server. The following code is the result:`package com.ibm.ws.eba.counter;`

```
public class CounterImpl implements Counter {  
  
    private int count = 0;
```

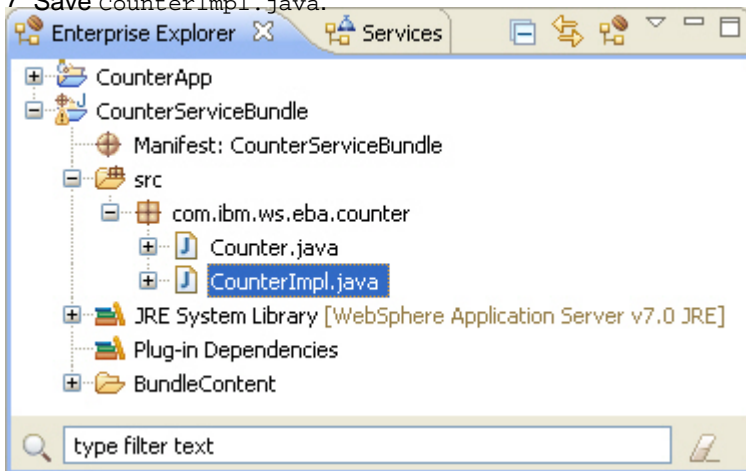
```

@Override
public int getCount() {
    return count++;
}

public void init() {
    System.out.println("CounterImpl.init() called.");
}
}

```

7. Save CounterImpl.java.



Exporting the package

By adding your package to the Export Packages list, you expose only this package to other clients outside of the bundle. All other packages are hidden from clients outside of the bundle. Use the Export Packages list to specify the name of any package that you want your bundle to export to the run time. If you do not specify the packages that are required by other bundles, the dependent bundles might not resolve.

1. Double-click **Manifest:CounterServiceBundle** to open the bundle manifest file in the editor.
2. Select the **Runtime** tab.
3. In the Exported Packages section of the editor, click **Add**. The Exported Packages dialog opens.
4. Click **com.ibm.ws.eba.counter** from the packages list and then click **OK**.
5. Save the bundle manifest file.

Lesson checkpoint

You created the business logic for the OSGi Counter Service application.

In this lesson, you learned about the following topics:

- How to create a Java package.
- How to create a Java interface file.
- How to create a Java method.
- How to create an implementation class.
- How to declare the packages that are visible outside of the bundle by using the Export-Package property in the bundle manifest file.

[< Previous](#) | [Next >](#)

Lesson 3: Create the blueprint configuration file

The blueprint configuration file contains the component assembly and configuration information for a bundle. The file describes how components are registered in the OSGi service registry or how they look up services from the OSGi service registry. This information is used at run time to instantiate and configure the required components when the bundle is started. In this tutorial, the blueprint file defines a service that other components can use to access the counter that is defined in Lesson 2.

In this lesson, you create the blueprint configuration file that defines and describes the services that are provided by the CounterServiceBundle.

To create the blueprint configuration file:

1. Right-click the project **CounterServiceBundle** and select **New > Blueprint File** and then click **Finish**. The blueprint configuration file opens in the editor.
2. Add the component assembly and configuration information to the blueprint configuration file:
 - A. In the Design tab of the editor, click **Add**. The Add Item dialog opens.
 - B. Click **Bean** and then click **OK**. The Bean Details dialog opens.
 - C. Configure the bean:
 1. In the **Bean ID** field, type `CounterBean`.
 2. In the **Bean Class** field, click **Browse**. The Type Selection dialog opens. In the **Choose type name** field, type `CounterImpl` and then select the **CounterImpl** class. Click **OK**.
 3. Click **OK** to accept the changes and close the dialog.
 4. In the editor, in the **Initialization method** field, under the Method References section, type `init`.
The bean is added to your blueprint file.
 - D. Click **Blueprint** and then click **Add**. The Add Item dialog opens.
 - E. Click **Service** and then click **OK**. The Service Details dialog opens.
 - F. Configure the service:
 1. In the **Service Interface** field, click **Browse** and then select the **Counter** interface that you created in [Lesson 2](#). Click **OK**.
 2. In the **Bean Reference** field, click **Browse** and then select **Bean: CounterBean**. Click **OK**.
 3. Click **OK** to accept the changes and close the dialog.
The service is added to your blueprint file.
3. Save the file.

Switch to the Source tab to view the blueprint configuration source:

```
<blueprint
xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <bean id="CounterBean" class="com.ibm.ws.eba.counter.CounterImpl" init-method="init"/>
  <service id="CounterBeanService" ref="CounterBean"
    interface="com.ibm.ws.eba.counter.Counter" />
</blueprint>
```

Tip: To format the source press Ctrl + Shift + F.

Learn more about this blueprint configuration file:

- bean

- The `bean` element defines the blueprint component that is instantiated.

- In this tutorial, the `bean` element results in the instantiation of component `Counter` by calling the `CounterImpl` class constructor. After the class is created, the initialization method `getCount()` is invoked.

- class

- The `class` attribute specifies which implementation class of the component is instantiated.

- id

- The `id` attribute identifies the component. It is mandatory if the component is referenced from elsewhere in the blueprint, for example if it is referenced from a service definition.

- init-method

- The `init-method` `init()` is called when the component is created. If you do not want to invoke a method during bundle initialization, remove this attribute.

- service

- The `service` element defines the export of a component to the OSGi service registry.

- In this tutorial, the `service` element exports the component with the name `Counter` as a service in the OSGi service registry with interface `com.ibm.ws.eba.counter.Counter`, specified by the `interface` attribute.

- ref

- The `ref` attribute refers to the component id of the exported component. This id is defined in the component element.

- interface

- The `interface` attribute refers to the interface that the component class implements.

For more information about the blueprint configuration file, see [OSGi Blueprint XML](#) and [OSGi Blueprint component model \(RFC124\)](#).

Lesson checkpoint

You created the blueprint configuration file for the OSGi Counter bundle.

In this lesson, you learned about the following topics:

- How to create a `blueprint.xml` file.
- How to configure a `blueprint.xml` file.

[< Previous](#) | [Next >](#)

Lesson 4: Create a servlet that accesses an OSGi service

In this lesson, you create a web enabled bundle to contain a servlet that accesses the counter service and displays the results of the service. The servlet accesses the OSGi service by looking up the service through a JNDI `InitialContext`. To create the servlet that accesses an OSGi service:

1. [Creating a Web enabled bundle.](#)
2. [Creating a servlet.](#)
3. [Adding required packages to the bundle manifest.](#)
4. [Creating the implementation to look up an OSGi service.](#)

Creating a Web enabled bundle

To create the web enabled bundle that contains the servlet:

1. Click **File > New > Other** and then expand **OSGi**.
2. Click **OSGi Bundle Project** and then click **Next**. The New OSGi Bundle Project opens.
3. In the **Project name** field, type `CounterWebBundle`.
4. Select one of the following servers from the Target runtime list:
 - **WebSphere Application Server v7.0**
 - **WebSphere Application Server v8.0**
 - **WebSphere Application Server v8.5**
 - **WebSphere Application Server V8.5 Liberty Profile**
5. In the Configuration section, click **Add Web support** and select **Web 2.5** from the web support list.
6. Ensure that `CounterApp` is displayed in the **Application project** field and then click **Finish**.

Creating a servlet

To create the servlet that accesses the counter service:

1. In Enterprise Explorer, right-click **CounterWebBundle** and then select **New > Servlet**. The Create Servlet wizard opens.
2. In the **Java package** field, type `com.ibm.ws.eba.servlet`.
3. In the **Class name** field, type `CounterServlet` and then click **Finish**. The servlet is created in the bundle and it opens in the editor.

Adding required packages to the bundle manifest

To add the required packages to the bundle manifest:

1. Double-click **Manifest: CounterWebBundle** to open it in the editor.
2. Switch to the Dependencies tab.
3. In the Imported Packages section, click **Add**. The Package Selection dialog opens.
4. In the **Exported Packages** field, type `com.ibm.ws.eba.counter`.
5. Select `com.ibm.ws.eba.counter` from the list and then click **OK**. The Imported Packages section looks similar to the

Imported Packages

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.

com.ibm.ws.eba.counter

45

Add...

Note: If your target runtime server is WebSphere Application Server V8.5 Liberty Profile, then you must also import the `javax.naming` package. Importing the `javax.naming` package prevents a `ClassNotFoundException` error when you run the application. The following steps describe how to import the `javax.naming` package:

- A. In the Imported Packages section, click **Add**. The Package Selection dialog opens.
- B. In the **Exported Packages** field, type `javax.naming`.
- C. Select `javax.naming` from the list and then click **OK**.

If you do not import the package, a `ClassNotFoundException` error occurs when you run the application on WebSphere Application Server with Liberty Profile. The error occurs because the application expects the package with the missing class to be provided by the server runtime instead of being provided as a user package.

6. Save the bundle manifest file.

Creating the implementation to look up an OSGi service

The following steps describe how to create the implementation for the `doGet()` method, which looks up an OSGi service by using `InitialContext`:

1. In Enterprise Explorer, double-click **CounterServlet.java** to open it in the editor.
2. Locate the `doGet()` method and add the following implementation:

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    Counter counter;
    try {
        InitialContext ic = new InitialContext();
        counter = (Counter) ic.lookup("osgi:service/"+Counter.class.getName());
        response.getOutputStream().println("counter="+counter.getCount());
    }
    catch (NamingException e) {
        e.printStackTrace(System.out);
    }
}
```

3. In the main menu click **Source > Organize Imports**. The Organize Imports dialog opens.
4. Select **com.ibm.ws.eba.counter.Counter** and then click **Finish**.
5. Save **CounterServlet.java**.

The servlet code looks similar to the following code:

```
import java.io.IOException;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.ws.eba.counter.Counter;

/**
 * Servlet implementation class CounterServlet
 */
public class CounterServlet extends HttpServlet {
```

```

private static final long serialVersionUID = 1L;

/**
 * Default constructor.
 */
public CounterServlet() {
    // TODO Auto-generated constructor stub
}

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    Counter counter;
    try {
        InitialContext ic = new InitialContext();
        counter = (Counter) ic.lookup("osgi:service/"+Counter.class.getName());
        response.getOutputStream().println("counter="+counter.getCount());
    }
    catch (NamingException e) {
        e.printStackTrace(System.out);
    }
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
}
}

```

Lesson checkpoint

You created the servlet that accesses an OSGi service by looking up the service through a JNDI `InitialContext`.

In this lesson, you learned about the following topics:

- How to create a web enabled OSGi bundle.
- How to create a servlet.
- How to import bundle dependencies by using Imported Packages.
- How to look up an OSGi service by using `InitialContext`.

[< Previous](#) | [Next >](#)

Lesson 5: Deploy the OSGi Counter application

Follow these steps to deploy the application to a WebSphere® Application Server installation that has the OSGi feature pack installed. Refer to the WebSphere Application Server documentation for installation and configuration details.

To deploy your OSGi Counter application to a server:

1. In Enterprise Explorer, expand **CounterWebBundle** > **CounterWebBundle** > **Servlets**.
2. Right click **CounterServlet** and select **Run As** > **Run on Server**. The Run On Server dialog opens.
3. Click **Finish**.

The web browser opens and the string `counter=0` is displayed. Each time the page is reloaded the value increments.

Switch to the Console view (**Window** > **Show View** > **Console**) to view the output from the server. A successful outcome displays the message

```
CounterImpl.init() called based on the initialization method entry for the CounterImpl bean in the blueprint file:[3/30/10 15:30:58:546 EDT] 0000004c StepStartBLA A CWWMH0300I: Starting business-level application "WebSphere:blaname=CounterApp".
```

```
[3/30/10 15:30:59:187 EDT] 0000004c webapp I com.ibm.ws.webcontainer.webapp.WebGroupImpl WebGroup SRVE0169I:
```

```
Loading Web Module: CounterWebBundle.
```

```
[3/30/10 15:30:59:265 EDT] 0000004c WASSessionCor I SessionContextRegistry getSessionContext SESN0176I: Will create a new session context for application key default_hostCounterWebBundle
```

```
[3/30/10 15:30:59:281 EDT] 0000004c webcontainer I com.ibm.ws.wshebcontainer.VirtualHost addWebApplication SRVE0250I:
```

```
Web Module CounterWebBundle has been bound to default_host[*:9083,*:80,*:9446,*:5067,*:5066,*:443].
```

```
[3/30/10 15:30:59:296 EDT] 0000004c FileLocatorIm E CWPST0164E: The CounterWebBundle composition unit is not found.
```

```
[3/30/10 15:30:59:312 EDT] 0000004c StepStartBLA A CWWMH0196I: Business-level application
```

```
"WebSphere:blaname=CounterApp" was started successfully.
```

```
[3/30/10 15:30:59:312 EDT] 00000016 SystemOut O CounterImpl.init() called
```

Note: If the output from the `CounterImpl.init()` is not displayed in the console output, check the output for error messages during deployment or startup of the application and then check the blueprint files for possible errors in the bean and service definitions.

Lesson checkpoint

You deployed the application to WebSphere Application Server.

In this lesson, you learned how to run your application on a server.

[< Previous](#) | [Next >](#)

Lesson 6: Add dependency injection within a bundle

Dependency injection allows one bean to access another bean without having to implement any code to create the bean instance. The required bean instance is created by the blueprint container by using information that is contained in the blueprint configuration file.

Before you begin this lesson, stop the server. In the Servers view (**Window > Show View > Servers**), right-click **WebSphere Application Server** and select **Stop**.

To add dependency injection within a bundle:

1. [Adding a service to the bundle](#)
2. [Updating the blueprint configuration file](#)
3. [Updating the servlet to invoke the new service](#)
4. [Deploying the application](#)

Adding a service to the bundle

1. Create the interface class:

- A. In Enterprise Explorer, expand **CounterServiceBundle > src**.
- B. Right-click **com.ibm.ws.eba.counter** and then select **New > Interface**. The New Java™ Interface wizard opens.
- C. In the **Name** field, type `Greet` and then click **Finish**. The interface is created in the package and it opens in the editor.
- D. Add the `getText()` prototype to the interface. The following code is the result:`package com.ibm.ws.eba.counter;`

```
public interface Greet {  
    public String getText();  
}
```

2. Create the implementation class:

- A. Right-click **com.ibm.ws.eba.counter** and then select **New > Class**. The New Java Class wizard opens.
- B. In the **Name** field, type `GreetImpl`.
- C. Click **Add**. The Implemented Interfaces Selection dialog opens.
- D. In the **Choose interfaces** field, type `Greet`. Select the **Greet** interface and then click **OK**.
- E. Click **Finish**. The class is created in the package and it opens in the editor.
- F. Replace the `GreetImpl` class with the following implementation:`package com.ibm.ws.eba.counter;`

```
public class GreetImpl implements Greet {  
  
    private Counter counter;  
  
    public void setCounter(Counter c) {  
        counter = c;  
    }  
  
    @Override  
    public String getText() {  
        return counter.getCount()+" Hello";  
    }  
}
```

```

}

public void init() {
    System.out.println("GreetImpl.init() called");
}

}

```

G. Save `GreetImpl.java`.

The private variable `counter` is the reference to an instance of the Counter service that is initialized by the container by using dependency injection. It is set by using the `setCounter` method.

Updating the blueprint configuration file

Carry out the following steps to configure the blueprint configuration file to start the Greet service and inject an instance of the Counter service:

1. In Enterprise Explorer, right-click **CounterServiceBundle** and then select **OSGi > Open Blueprint File** to open the blueprint file in the editor.
2. Add the component assembly and configuration information to the blueprint configuration file:
 - A. In the Design tab of the editor, select **Blueprint** and then click **Add**. The Add Item dialog opens.
 - B. Click **Bean** and then click **OK**. The Bean Details dialog opens.
 - C. Configure the bean:
 1. In the **Bean ID** field, type `GreetBean`.
 2. In the **Class** field, click **Browse**. The Type Selection dialog opens. In the **Choose type name** field, type `GreetImpl` and then select the **GreetImpl** class. Click **OK**.
 3. Click **OK** to accept the changes and close the dialog.
 4. In the editor, in the **Initialization method** field, type `init`.
The bean is added to your blueprint file.
 - D. In the Overview section, click **GreetBean (Bean)** and then click **Add**. The Add Item dialog opens.
 - E. Click **Property** and then click **OK**.
 - F. Configure the Property:
 - In the **Name** field, type `counter`.
 - In the **Reference** field, click **Browse**. The Reference Selection dialog opens. Click **Bean: CounterBean** and then click **OK**.
 - G. In the Overview section, click **Blueprint** and then click **Add**.
 - H. Click **Service** and then click **OK**. The Service Details dialog opens.
 - I. Configure the service:
 1. In the **Service Interface** field, click **Browse** and then select the **Greet** interface.
 2. In the **Bean Reference** field, click **Browse** and then select **Bean: GreetBean**. Click **OK**.
 3. Click **OK** to accept the changes and close the dialog.
The service is added to your configuration.
3. Save the file.

Updating the servlet to invoke the new service

1. In Enterprise Explorer, expand **CounterWebBundle > Java Resources > src > com.ibm.ws.eba.servlet**.
2. Double-click **CounterServlet.java** to open it in the editor.
3. Locate the `doGet()` method and replace it with the following implementation:

```

protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {

```

```

Greet greet;

try {

    InitialContext ic = new InitialContext();

    greet = (Greet) ic.lookup("osgi:service/"+Greet.class.getName());

    response.getOutputStream().println("greet.getText()="+greet.getText());

} catch (NamingException e) {

    e.printStackTrace(System.out);

}

}

```

4. In the main menu click **Source > Organize Imports**. The import statements are updated.
5. Save CounterServlet.java.

Deploying the application

1. In Enterprise Explorer, expand **CounterWebBundle > CounterWebBundle > Servlets**.
2. Right click **CounterServlet** and select **Run As > Run on Server**. The Run On Server dialog opens.
3. Click **Finish**.

The string `greet.getText()=0 Hello` is displayed by the browser. Each time the page is reloaded the value increments. Switch to the Console view (**Window > Show View > Console**) to view the output from the server. A successful outcome displays the output from `CounterImpl.init()` and `GreetImpl.init()`, based on the initialization method entries for the `CounterImpl` and `GreetImpl` beans in the blueprint file: [3/30/10 16:59:41:734 EDT] 00000072 StepStartBLA A CWWMH0300I: Starting business-level application "WebSphere:blaname=CounterApp".

```

[3/30/10 16:59:42:406 EDT] 00000072 webapp      I com.ibm.ws.webcontainer.webapp.WebGroupImpl WebGroup SRVE0169I:
Loading Web Module: CounterWebBundle.

[3/30/10 16:59:42:453 EDT] 00000072 WASSessionCor I SessionContextRegistry getSessionContext SESN0176I: Will create a new
session context for application key default_hostCounterWebBundle

[3/30/10 16:59:42:468 EDT] 00000072 webcontainer  I com.ibm.ws.wshebcontainer.VirtualHost addWebApplication SRVE0250I:
Web Module CounterWebBundle has been bound to default_host[*:9083,*:80,*:9446,*:5067,*:5066,*:443].

[3/30/10 16:59:42:468 EDT] 00000072 FileLocatorIm E   CWPST0164E: The CounterWebBundle composition unit is not found.

[3/30/10 16:59:42:500 EDT] 00000072 StepStartBLA A   CWWMH0196I: Business-level application
"WebSphere:blaname=CounterApp" was started successfully.

[3/30/10 16:59:42:500 EDT] 00000016 SystemOut    O CounterImpl.init() called

[3/30/10 16:59:42:500 EDT] 00000016 SystemOut    O GreetImpl.init() called

```

Note: If the output from the `CounterImpl.init()` and `GreetImpl.init()` is not displayed in the console output, check the output for error messages during deployment or startup of the application and then check the blueprint files for possible errors in the bean and service definitions.

Lesson checkpoint

You learned how to use blueprint dependency injection to allow one bean to use the services of another.

In this lesson, you learned about the following topics:

- How to write the code for a bean that uses another bean.
- How to define property in the blueprint configuration file that instructs the blueprint container to initialize the variable by using dependency injection.

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 7: Add dependency injection between bundles

In the previous lesson, you learned how to use dependency injection within a bundle. Dependency injection between bundles requires an extra reference entry in the blueprint configuration file.

Before you begin this lesson, stop the server. In the Servers view (**Window > Show View > Servers**), right-click **WebSphere Application Server** and select **Stop**.

To add dependency injection between bundles:

1. [Creating a bundle project](#)
2. [Adding a service to the bundle](#)
3. [Updating the bean to use the new service](#)
4. [Updating the blueprint configuration file](#)
5. [Deploying the application](#)

Creating a bundle project

To create a bundle project and add it to an OSGi application project:

1. Click **File > New > Other** and then expand **OSGi**.
2. Click **OSGi Bundle Project** and then click **Next**. The New OSGi Bundle Project opens.
3. In the **Project name** field, type `CounterWorldBundle`.
4. Select one of the following servers from the Target runtime list:
 - **WebSphere Application Server v7.0**
 - **WebSphere Application Server v8.0**
 - **WebSphere Application Server v8.5**
 - **WebSphere Application Server V8.5 Liberty Profile**
5. Ensure that `CounterApp` is displayed in the **Application project** field and then click **Finish**.

Adding a service to the bundle

In this lesson, you create a simple service, `World` that returns a text string from the method `World.getText()`.

1. Create the package:
 - A. In Enterprise Explorer, expand **CounterWorldBundle**.
 - B. Right-click **src** and then select **New > Package**. The New Java™ Package wizard opens.
 - C. In the **Name** field, type `com.ibm.ws.eba.world` and then click **Finish**. The package is created in the `src` folder.
2. Create the interface class:
 - A. Right-click the package **com.ibm.ws.eba.world** and then click **New > Interface**. The New Java Interface wizard opens.
 - B. In the **Name** field, type `World` and then click **Finish**. The interface is created in the package and it opens in the editor.
 - C. Add the `getText()` method to the interface. The following code is the result:`package com.ibm.ws.eba.world;`

```
public interface World {  
    public String getText();  
}
```

D. Save `World.java`.

3. Create the implementation class:

A. Right-click the package **com.ibm.ws.eba.world** and then click **New > Class**. The New Java Class wizard opens.

B. In the **Name** field, type `WorldImpl`.

C. Click **Add**. The Implemented Interfaces Selection dialog opens.

D. In the **Choose interfaces** field, type `World`. Select the **World** interface for `com.ibm.ws.eba.world` and then click **OK**.

E. Click **Finish**. The class is created in the package and it opens in the editor.

F. Add the implementation for the `getText()` method. Add an initialization method that confirms that the service starts on the server. The following code is the result:

```
package com.ibm.ws.eba.world;

public class WorldImpl implements World {

    @Override
    public String getText() {
        return " World!";
    }

    public void init() {
        System.out.println("WorldImpl.init() called.");
    }
}
```

G. Save `WorldImpl.java`.

4. Create the blueprint configuration file:

A. Right-click the project **CounterWorldBundle** and select **New > Other > OSGI > Blueprint File** and then click **Next**.

B. Click **Finish**. The blueprint configuration file opens in the editor.

5. Add the component assembly and configuration information to the blueprint configuration file:

A. In the Design tab of the editor, click **Add**. The Add Item dialog opens.

B. Click **Bean** and then click **OK**. The bean is added to your configuration.

C. Configure the bean:

- In the **ID** field, type `WorldBean`.

- In the **Class** field, click **Browse**. The Type Selection dialog opens. In the **Choose type** name field, type `WorldImpl` and then select the **WorldImpl** class. Click **OK**.

- In the **Initialization method** field, type `init`.

D. Click **Blueprint** and then click **Add**. The Add Item dialog opens.

E. Click **Service** and then click **OK**. The service is added to your configuration.

F. Configure the service:

- In the **ID** field, type `WorldService`.

- In the **Interface** field, click **Browse** and then select the **World** interface.

- In the **Reference** field, click **Browse** and then select **Bean: WorldBean**. Click **OK**.

G. Save the file.

6. Export the package:

A. Expand **CounterWorldBundle** and then double-click **Manifest: CounterWorldBundle** to open the bundle manifest file in the editor.

B. Select the **Runtime** tab.

C. In the Exported Packages section of the editor, click **Add**. The Exported Packages dialog opens.

D. Click **com.ibm.ws.eba.world** from the packages list and then click **OK**.

E. Save the bundle manifest file.

Updating the bean to use the new service

Now that you have implemented the World service, you need to update the GreetImpl bean and the CounterServiceBundle bundle manifest file to use the World service.

1. Import the package:

- A. In Enterprise Explorer, expand **CounterServiceBundle** and then double-click **Manifest: CounterServiceBundle** to open the bundle manifest file in the editor.
- B. Select the **Dependencies** tab.
- C. In the Imported Packages section of the editor, click **Add**. The Imported Packages dialog opens.
- D. Click **com.ibm.ws.eba.world** from the packages list and then click **OK**.
- E. Save the bundle manifest file.

2. Update GreetImpl.java:

- A. In Enterprise Explorer, expand **CounterServiceBundle > src > com.ibm.ws.eba.counter** and then double-click **GreetImpl.java** to open the file in the editor.

- B. Add the following variable declaration and set method for the World bean:

```
private World worldBean;

public void setWorldBean(World b)
{
    worldBean = b;
}
```

- C. In the main menu click **Source > Organize Imports**. The missing import statements are added to your code.

- D. Update the `getText()` method to call the new service:

```
public String getText(){
    return counter.getCount()+" Hello"+worldBean.getText();
}
```

- E. Save the file.

The following code is the result:

```
package com.ibm.ws.eba.counter;

import com.ibm.ws.eba.world.World;

public class GreetImpl implements Greet {
    private World worldBean;

    public void setWorldBean(World b)
    {
        worldBean = b;
    }

    private Counter counter;

    public void setCounter(Counter c) {
        counter = c;
    }

    @Override
    public String getText(){
        return counter.getCount()+" Hello"+worldBean.getText();
    }

    public void init() {
```



```

        System.out.println("GreetImpl.init() called");
    }

}

```

Updating the blueprint configuration file

In order for the blueprint container to inject an instance of the World bean into GreetImpl, you need to add the reference to the blueprint configuration file.

1. Expand **CounterServiceBundle > BundleContent > OSGI-INF > blueprint** and then double-click **blueprint.xml** to open the blueprint configuration file in the editor.
2. In the Design tab of the editor, select **Blueprint** and then click **Add**. The Add Item dialog opens.
3. Click **Reference** and then click **OK**. The Reference Details dialog opens.
4. Configure the Reference:
 - In the **Reference ID** field, type `WorldRef`.
 - In the **Reference Interface** field, click **Browse** and then select the **World** interface that you created. Click **OK**.
 - Click **OK** to accept the changes and close the dialog.

The reference is added to your blueprint file.
5. In the Overview section, click the **GreetBean (Bean)** node and then click **Add**. The Add Item dialog opens.
6. Click **Property** and then click **OK**. The property is added to your configuration.
7. Configure the property:
 - In the **Name** field, type `worldBean`.
 - In the **Reference** field, click **Browse**. The Reference Selection dialog opens. Click **Reference:WorldRef** and then click **OK**.
8. Save the file.

Deploying the application

1. In Enterprise Explorer, expand **CounterWebBundle > CounterWebBundle > Servlets**.
2. Right click **CounterServlet** and select **Run As > Run on Server**. The Run On Server dialog opens.
3. Click **Finish**.

The string `greet.getText()=0 Hello World!` is displayed in the browser. Each time the page is reloaded the value increments.

Switch to the Console view (**Window > Show View > Console**) to view the output from the server. A successful outcome displays the output from `CounterImpl.init()`, `GreetImpl.init()`, and `WorldImpl.init()`, based on the initialization method entries for the `CounterImpl`, `GreetImpl`, and `WorldImpl` beans in the blueprint file:

```

1/10 13:07:26:250 EDT] 000000aa
StepStartBLA A CWWMH0300I: Starting business-level application "WebSphere:blaname=CounterApp".

[3/31/10 13:07:27:000 EDT] 000000aa webapp I com.ibm.ws.webcontainer.webapp.WebGroupImpl WebGroup SRVE0169I:
Loading Web Module: CounterWebBundle.

[3/31/10 13:07:27:046 EDT] 000000aa WASSessionCor I SessionContextRegistry getSessionContext SESN0176I: Will create a new
session context for application key default_hostCounterWebBundle

[3/31/10 13:07:27:062 EDT] 000000aa webcontainer I com.ibm.ws.wshebcontainer.VirtualHost addWebApplication SRVE0250I:
Web Module CounterWebBundle has been bound to default_host[*:9083,*:80,*:9446,*:5067,*:5066,*:443].

[3/31/10 13:07:27:078 EDT] 000000aa FileLocatorIm E CWPST0164E: The CounterWebBundle composition unit is not found.

[3/31/10 13:07:27:093 EDT] 000000aa StepStartBLA A CWWMH0196I: Business-level application
"WebSphere:blaname=CounterApp" was started successfully.

[3/31/10 13:07:27:109 EDT] 00000066 SystemOut O WorldImpl.init() called

[3/31/10 13:07:27:109 EDT] 00000015 SystemOut O CounterImpl.init() called

```

```
[3/31/10 13:07:27:125 EDT] 00000015 SystemOut      O GreetImpl.init() called
```

Note: If the output from the `CounterImpl.init()`, `GreetImpl.init()`, and `WorldImpl.init()` does not display in the console output, check the output for error messages during deployment or startup of the application and then check the blueprint files for possible errors in the bean and service definitions.

Lesson checkpoint

You learned how to use blueprint dependency injection to allow one bean to use the services of another.

In this lesson, you learned how to use a blueprint Reference to configure cross bundle dependency injection.

[< Previous](#) | [Next >](#)

Summary

You created an OSGi application that consists of an OSGi web bundle. The OSGi web bundle contains a servlet that accesses a service that is provided in another bundle project.

Lessons learned

In this tutorial, you learned how to complete the following tasks:

- Create an OSGi bundle and an OSGi application.
- Create and configure an OSGi blueprint configuration file.
- Configure component dependencies.
- Add dependency injection within a bundle.
- Add dependency injection between bundles.
- Deploy an OSGi application to a server.

Additional resources

For more information about developing OSGi applications, see the online help by clicking **Help > Help Contents**. For more in-depth technical articles on OSGi applications, see [developerWorks®](#).

Web tutorials

The Web tutorials provide step by step instructions for you to follow, to complete specific tasks relating to Web applications.

The Web tutorials are contained in the following subsections:

- **[Tutorial: Create a loan payment calculator with Dojo](#)**

Learn how to create a Dojo-based loan payment calculator.

Parent topic: [Tutorials](#)

Tutorial: Create a loan payment calculator with Dojo

Learn how to create a Dojo-based loan payment calculator.

Learning objectives

The calculator accepts three pieces of input: loan amount, interest rate, and term. It outputs the monthly payment, a pie chart displaying the percentage of loan costs going towards principal and interest, and an amortization table. There is no submit button because the output fields are updated in real time as users enter or change input. In this tutorial you learn how to:

- Create a Dojo-enabled web application to contain all the resources required by your application.
- Create a user interface with these Dojo widgets:
 - A Dojo layout widget to define the placement of objects on the web page
 - A Dojo custom widget to calculate and display monthly payments
 - A pie chart to display a breakdown of loan costs
- Use content assist, templates, and wizards to write Dojo code and HTML
- Deploy your project to a server
- Create a custom Dojo build to optimize your application
- Debug your web application using Firebug

Time required

This tutorial should take approximately 30 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete. You can also import the completed solution for the tutorial found at the end of [lesson 6](#) .

[< Previous](#) | [Next >](#)

Introduction: Create a loan payment calculator with Dojo

Learn how to create a Dojo-based loan payment calculator.

Learning objectives

In this tutorial you will:

- Create a Dojo-enabled web application to contain all the resources required by your application.
- Create a user interface with these Dojo widgets:
 - A Dojo layout widget to define the placement of objects on the web page
 - A Dojo custom widget to calculate and display monthly payments
 - A pie chart to display a breakdown of loan costs
- Use content assist, templates, and wizards to write Dojo code and HTML
- Deploy your project to a server
- Create a custom Dojo build to optimize your application
- Debug your web application using Firebug

Time required

This tutorial should take approximately 30 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete. You can also import the completed solution for the tutorial found at the end of [lesson 6](#).

Prerequisites

1. Install the Web Development Tools and Dojo Toolkit.
2. Install Mozilla Firefox for lessons 4 and 6.

Lessons in this tutorial

- [Lesson 1: Create a Dojo-enabled web project](#)

In this lesson, you learn how to create a web project that is configured for Dojo application development by enabling specific project features.

- [Lesson 2: Create a custom Dojo widget](#)

The Dojo toolkit includes dozens of standard widgets, including input fields, combo boxes and radio buttons. You can also create custom widgets to encapsulate reusable UI elements or a specific piece of functionality. In this lesson you create a new custom Dojo widget.

- [Lesson 3: Add the custom Dojo widget to a web page](#)

In this lesson you insert your custom Dojo widget into a web page that is laid out using a Dojo layout widget. You use Dojo APIs to connect your widget to the output field and display the results.

- [Lesson 4: Add a pie chart using the Dojo charting API \(optional\)](#)

In this optional lesson, you use content assist and the Dojo charting API to add a pie chart to your results page. The pie chart displays the percentage of total loan costs going towards interest and principal.

- [Lesson 5: Create a Dojo custom build \(optional\)](#)

This optional lesson highlights the steps required to create a Dojo custom build. The purpose of a custom Dojo build is to create an efficient version of Dojo, and of your code, that is suitable for deployment.

- [Lesson 6: Debugging your Dojo application with Firebug \(optional\)](#)

This optional lesson highlights how to use Firebug to debug your web application.

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 1: Create a Dojo-enabled web project

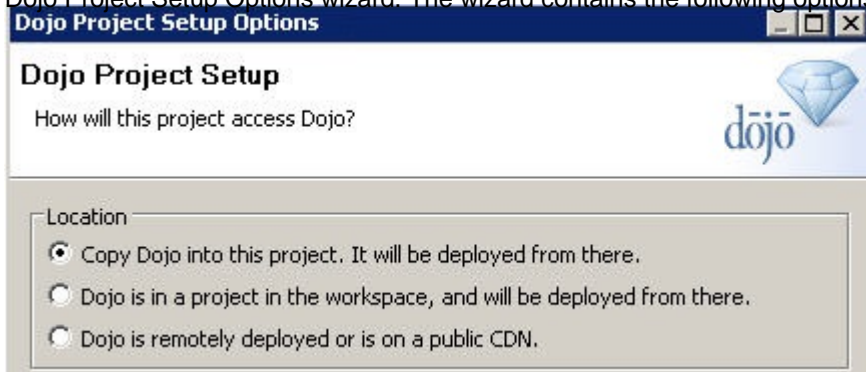
In this lesson, you learn how to create a web project that is configured for Dojo application development by enabling specific project features.

This product provides a project setup wizard where you can customize how your application accesses Dojo during development and at runtime. The simplest option (and the one used in this tutorial) is to have a copy of Dojo in your project and deploy it along with the rest of your project resources. Another option is to use a public Content Delivery Network (CDN), a remote server with a copy of Dojo. The use of CDNs are not covered by this tutorial, but they are simple to use.

For more information about CDNs, see www.dojotoolkit.org/download/.

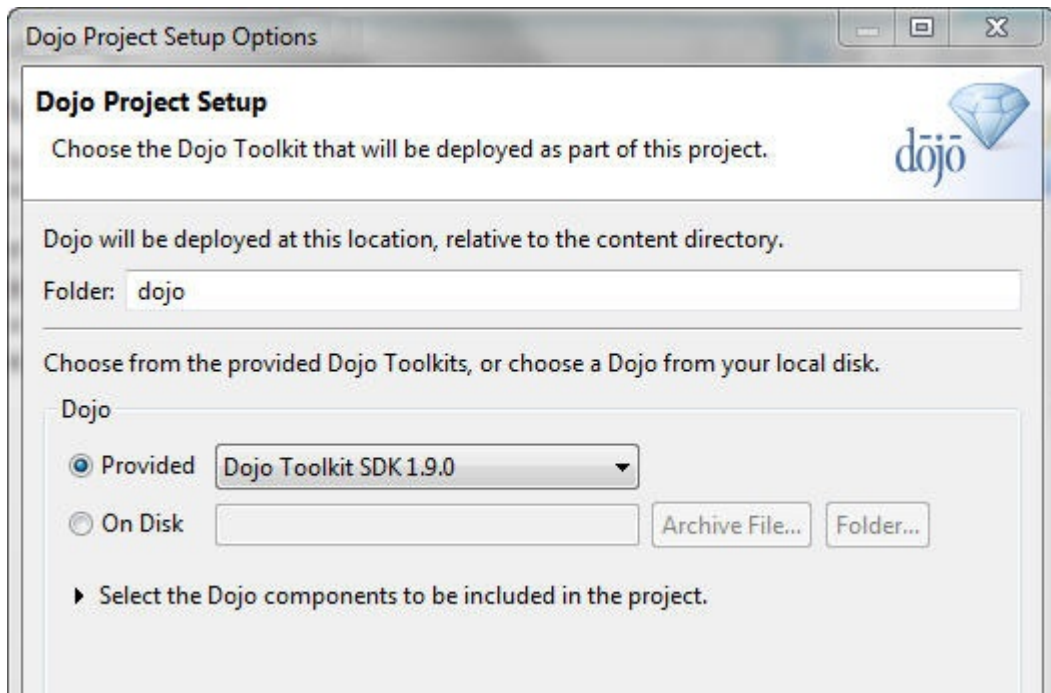
To create a web project that is configured for Dojo application development:

1. In the main menu, click **File > New > Web Project**.
2. In the **Name** field, type `LoanPaymentCalculator`.
3. From the list of project templates, click **Dojo Toolkit** to use the Dojo Toolkit project template to create your web project.
4. In the Programming Model section, click **Client-side only (HTML, JavaScript,...)** to use the Client-side only programming model when creating your project.
5. Click **Next** to configure your new web project.
6. From the list of available configuration options, click **Dojo Toolkit** to open the Dojo Toolkit page. The Dojo Toolkit included in this product includes additional IBM® extensions to the base Dojo Toolkit, including libraries for ATOM (ATOM Syndication Format) data access, analog and bar gauges, and simplified access for SOAP Web services. By default, the latest Dojo Toolkit supported by IBM is copied into your web project.
7. To determine how you want to use the Dojo Toolkit in your web project, click **Change these setup options** to open the Dojo Project Setup Options wizard. The wizard contains the following options:



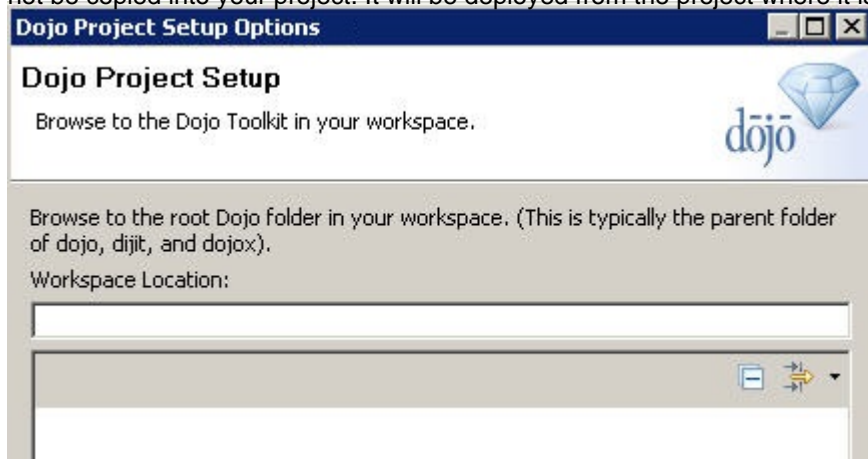
- Copy Dojo into this project. It will be deployed from there.

- The Dojo toolkit is included inside your web project. You can specify the name of the Dojo folder and whether to use the default Dojo distribution included in the runtime environment or to use a Dojo distribution from disk.



- Dojo is in a project in the workspace and will be deployed from there.

- On this page you can browse to the root Dojo folder in another project in your workspace. The copy of Dojo will not be copied into your project. It will be deployed from the project where it is currently located.



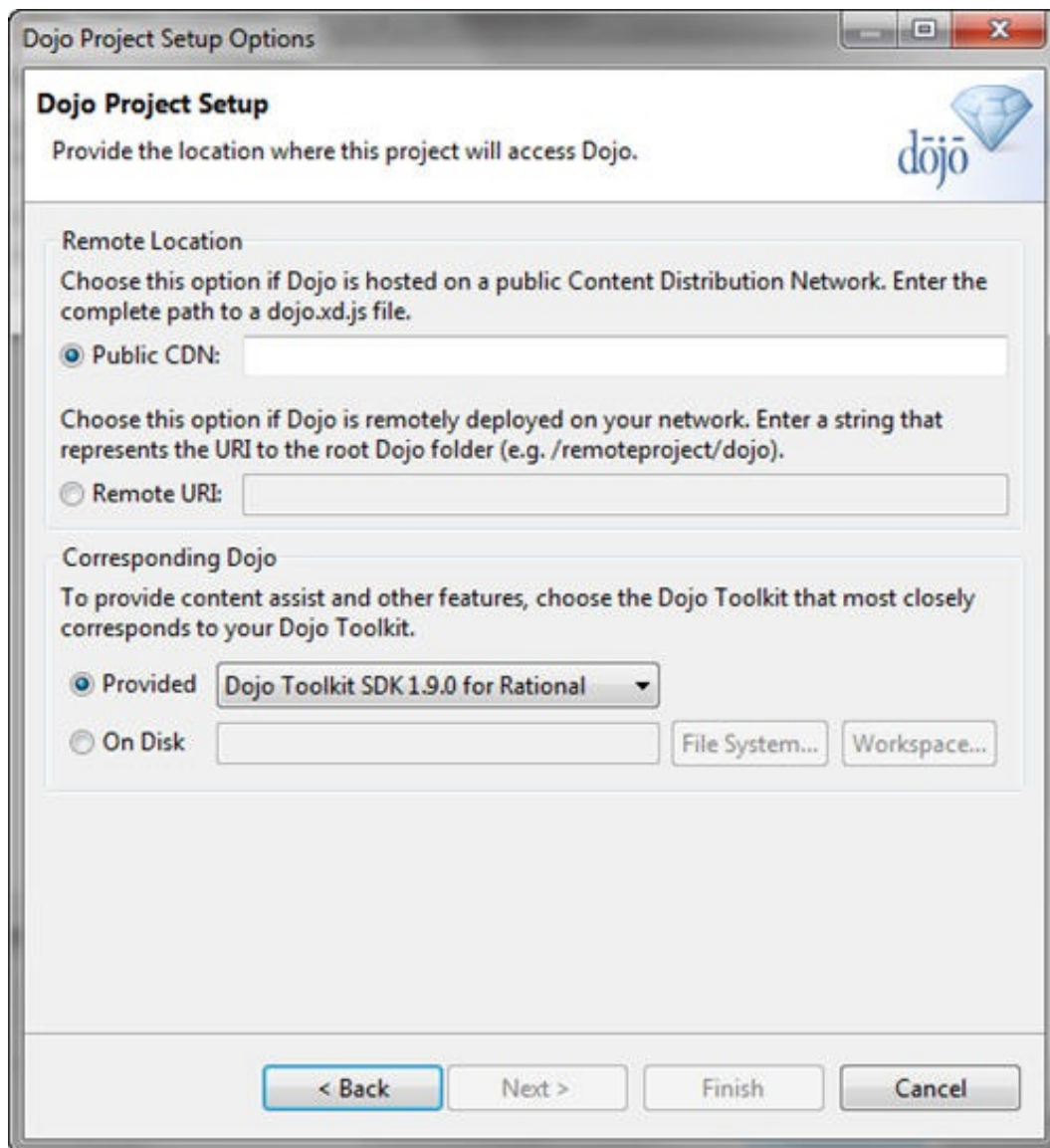
- Dojo is remotely deployed or is on a public CDN.

- Use this option if your application uses a remotely hosted public CDN (content delivery network) or an existing copy of Dojo already deployed on your network.

- **Public CDN:** You can enter the URL of a publicly available content delivery network. Content delivery networks provide geographically distributed hosting for open source JavaScript libraries. When a browser resolves the URL in your web application, the browser downloads the file from the closest available server.

- **Remote URI:** You can enter the URI of the remote location to the root Dojo folder.

In the Corresponding Dojo section, you can choose a Dojo source distribution that is the closest match to your remote Dojo Toolkit. If Dojo is not contained in your project the tools must reference a corresponding copy of Dojo in order to provide content assist and validation. You can choose the default Dojo Toolkit provided with this product, or browse to a Dojo folder in your workspace or file system. This option does not copy Dojo into your project or workspace.



8. Click **Copy Dojo into this project. It will be deployed from there.** to include the Dojo Toolkit inside your web project for this tutorial.
9. Click **Next** and accept the default. Click **Finish**.
10. Click **Finish** to create your web project.

Your project is now created and appears in the Enterprise Explorer view. Expand the WebContent folder to show the Dojo folder that contains all of the Dojo resources. If asked to switch to the Web Perspective, click **Yes**.

Lesson checkpoint

You created the Dojo-enabled web project.

You learned :

- How to create a Dojo-enabled web project.
- How to change the setup options for a Dojo-enabled web project.

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 2: Create a custom Dojo widget

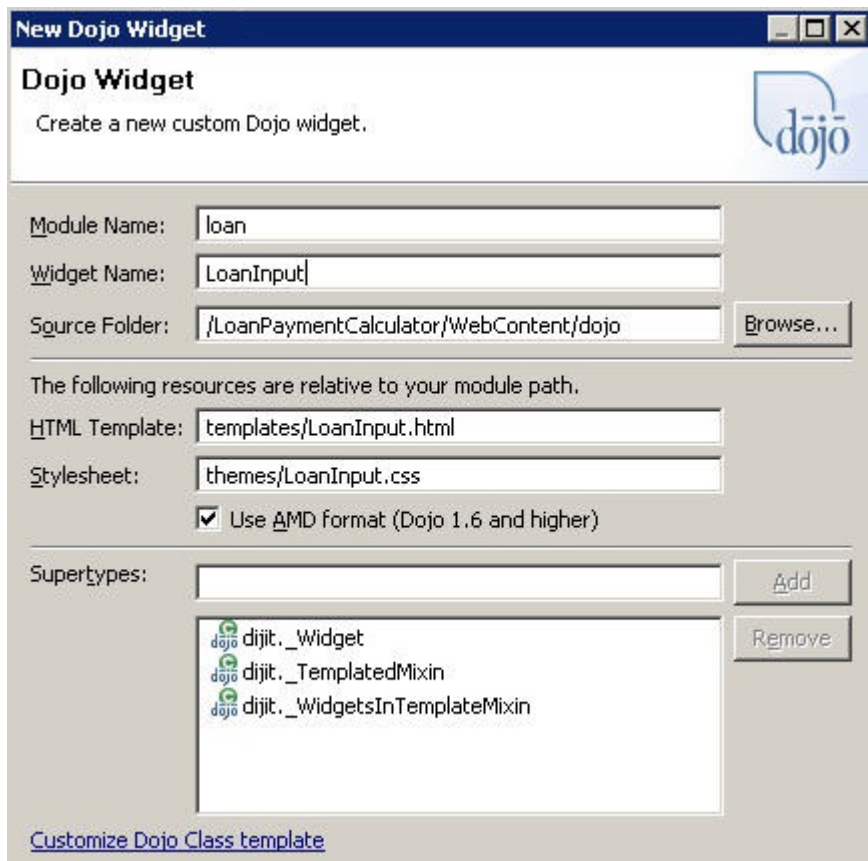
The Dojo toolkit includes dozens of standard widgets, including input fields, combo boxes and radio buttons. You can also create custom widgets to encapsulate reusable UI elements or a specific piece of functionality. In this lesson you create a new custom Dojo widget.

Dojo widgets are composed of three files that the New Dojo Widget wizard creates for you:

- An HTML file
- A JavaScript file
- A CSS file

You then edit the HTML template and the JavaScript file.

1. In the Enterprise Explorer view, right-click the WebContent/dojo folder and select **New > Dojo Widget**. The New Dojo Widget wizard appears.
2. In the **Module Name** field, enter `loan`.
3. In the **Widget Name** field, enter `LoanInput`. The HTML template and style sheet relative for the widget paths are populated automatically.



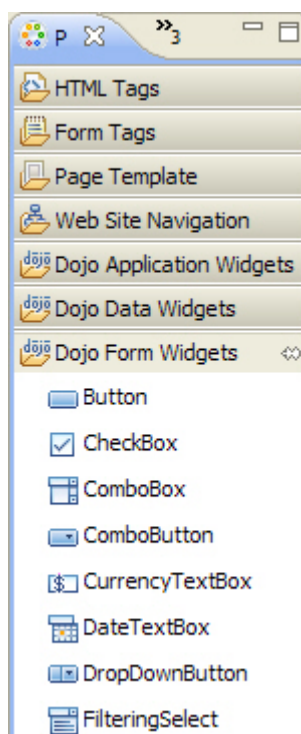
4. Click **Finish**. Three files are created under a folder named `dojo/loan`:
 - **templates/LoanInput.html**
 - The UI template for the widget.
 - **themes/LoanInput.css**
 - Provides the styling for the widget.
 - **LoanInput.js**
 - Provides the JavaScript backend and business logic portion of the widget.

5. The LoanInput JavaScript source file for the widget opens in the editor.
6. Under the `templateString` field, add three additional fields that will be used to hold the results of our calculation – `principal`, `interestPaid`, and `monthlyPayment` - they should all have default values of 0. Ensure that you add a comma after each field.


```
templateString : dojo.cache("loan", "templates/LoanInput.html"),
principal: 0,
interestPaid: 0,
monthlyPayment: 0,
```
7. Under the `postCreate` function, add a new function named `calculate`. Ensure that you add a comma after the `postCreate` function. Leave the new `calculate` function empty for now.


```
postCreate : function() {
},
// this function will perform the calculations for our loan payment
calculate: function() {
}
```
8. In the Enterprise Explorer view, double-click **templates/LoanInput.html** to open the HTML template for the widget.
9. Click **Next** and accept the defaults. Click **Finish**.
10. Within the existing div create three additional child div tags. You can use content assist by typing `<d` and then pressing `Ctrl + Space`. In the pop-up window, select **div** to insert the tag.
11. Within each new div tag add a text label – `Loan Amount:`, `Interest Rate:`, and `Term (years):`. When complete your code should look like this:


```
<div class="LoanInput">
<div>Loan Amount: </div>
<div>Interest Rate: </div>
<div>Term (years): </div>
</div>
```
12. Now add Dojo widgets for each of the input fields:
 - A. In the workspace, surface the palette by clicking the appropriate tab. You should see several drawers containing Dojo widgets.
 - B. Expand the **Dojo Form Widgets** drawer by clicking on it.



- C. Select the **CurrencyTextBox** and drop it next to the Loan Amount label inside your div tag.

D. Within the newly created input element, type `data-` and use content assist (Ctrl + Space) to show a list of attributes. Click **data-dojo-props** to insert it. To follow HTML5 standards, beginning in Dojo Toolkit SDK 1.7 attributes are set by default through `data-dojo-props`.

E. Inside the `data-dojo-props` attribute, type `cu` and use content assist (Ctrl + Space) to show a list of attributes. Click **currency** to insert it.

F. Inside the `currency` attribute, type `USD`. Your code should look like this:

```
<div>Loan Amount: <input type="text"
  data-dojo-type="dijit.form.CurrencyTextBox"
  data-dojo-props="currency: 'USD'">
</div>
```

G. Next, insert the Dojo widget markup for the Interest Rate field. Place your cursor inside the second `div` tag after the label, type `<input d>` and with your cursor next to the `d`, use content assist (Ctrl + Space) to show a list of attributes. Click **data-dojo-type** to insert it.

H. Inside the `data-dojo-type` attribute, invoke content assist again to show a list of available dojo widgets. Begin typing `dijit.form.N` until you see `NumberSpinner`. Click **NumberSpinner** to insert it into your page.

I. Add the `data-dojo-props` attribute. Inside the attribute, set the following properties, separated by commas:

1. `value : 5`
2. `smallDelta : .05`
3. `intermediateChanges : true`
4. `constraints : {min: 0}`

You can use content assist to insert these properties the same way you added the `currency` attribute previously.

When you are finished setting the properties for the attribute, your code should look like this:

```
<div>Interest Rate:
<input data-dojo-type="dijit.form.NumberSpinner"
  data-dojo-props="value: 5, smallDelta:= 0.05, intermediateChanges: true, constraints: {min: 0}"> </input></div>
```

J. From the palette drop a **ComboBox** widget into the Term (years) `div`.

K. Additional configuration for some widgets is available when you drop them from the palette, such as the `ComboBox`. In the Insert Combo Box dialog box you can add values for your `ComboBox`. Add values for 1, 2, 3, 4, 5, 10, 15, 30.

L. Set 15 as the default value by setting it to **True**. Click **OK**.

13. Next, add the `data-dojo-attach-point` and `data-dojo-attach-event` attributes to each of your input widgets. The value specified for the `data-dojo-attach-point` attribute is the name that the widget instance can be referenced by from the `LoanInput.js` file. The `data-dojo-attach-event` attribute adds event handling to the widgets.

A. Use content assist to add a `data-dojo-attach-point` attribute to each widget. Name them `amount`, `rate`, and `term` respectively.

B. Add `data-dojo-attach-event="onChange: calculate"` for each widget. Each time that an `onChange` event takes place on this widget it calls the `calculate` function that you added to the `LoanInput.js` file. The final result of your `LoanInput.html` file should look like this:

```
<div class="LoanInput">
  <div>Loan Amount: <input type="text"
    data-dojo-type="dijit.form.CurrencyTextBox"
    data-dojo-props="currency: 'USD'"
    data-dojo-attach-point="amount"
    data-dojo-attach-event="onChange: calculate"></div>
  <div>Interest Rate: <input data-dojo-type="dijit.form.NumberSpinner"
    data-dojo-props="value:5,
    smallDelta:0.05,
    intermediateChanges:true,
    constraints:{min: 0}"
    data-dojo-attach-point="rate"
    data-dojo-attach-event="onChange: calculate">
</div>
```

```

<div>Term (years): <select name="select"
data-dojo-type="dijit.form.ComboBox"
data-dojo-props="autocomplete:false"
data-dojo-attach-point ="term"
data-dojo-attach-event="onChange: calculate">
<option>1</option>
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
<option>10</option>
<option selected="selected">15</option>
<option>30</option>
</select>
</div>
</div>

```

14. Save and close **LoanInput.html** and reopen **LoanInput.js**.

15. Add Dojo module dependencies for the three widgets used in the html file. These dependencies will load the necessary resources to create the widgets when the page is run. The second argument of the `define` function is an array of dependencies. Directly after the `"dijit/_WidgetsInTemplateMixin"`, dependency, add the following module paths separated by commas:

- A. `"dijit/form/CurrencyTextBox"`
- B. `"dijit/form/NumberSpinner"`
- C. `"dijit/form/ComboBox"`

Your code should look like this:

```

define("loan/LoanInput",
[ "dojo", "dijit", "dijit/_Widget",
"dijit/_TemplatedMixin",
"dijit/_WidgetsInTemplateMixin",
"dijit/form/CurrencyTextBox",
"dijit/form/NumberSpinner",
"dijit/form/ComboBox",
"dojo/text!loan/templates/LoanInput.html"
],

```

16. Now add the following code for the `calculate` function that you created earlier in Step 7. You can experiment with content assist if you want. Note that standard JavaScript objects such as the `Math` object are available in content assist. The variables we defined earlier, `principal`, `interestPaid`, and `monthlyPayment` are all available as well.//

```

this function will perform the calculations for our loan repayment
calculate: function() {
  this.principal = this.amount.attr('value');
  if(this.principal == NaN) {
    this.monthlyPayment = 0;
    this.principal = 0;
    this.interestPaid = 0;
  } else {
    var interestRate = this.rate.attr('value') / 1200;
    var termInMonths = this.term.attr('value') * 12;

```

```

    this.monthlyPayment = Math.pow(1 + interestRate, termInMonths) - 1;
    this.monthlyPayment = interestRate + (interestRate / this.monthlyPayment);
    this.monthlyPayment = this.monthlyPayment * this.principal;

    this.interestPaid = (this.monthlyPayment * termInMonths) - this.principal;
}
}

```

The calculate function stores the principal of the loan, computes the monthly payment, and the amount of interest paid.

17. Save and close all of the files that make up the custom widget.

LoanInput.js should now look like this:

```

define("loan/LoanInput", [ "dojo", "dijit", "dijit/_Widget",
    "dijit/_TemplatedMixin", "dijit/_WidgetsInTemplateMixin", "dijit/form/CurrencyTextBox", "dijit/form/NumberSpinner",
    "dijit/form/ComboBox",
    "dojo/text!loan/templates/LoanInput.html" ], function(dojo, dijit,
    _Widget, _TemplatedMixin, _WidgetsInTemplateMixin, CurrencyTextBox, NumberSpinner, ComboBox) {
dojo.declare("loan.LoanInput", [ dijit._Widget, dijit._TemplatedMixin,
    dijit._WidgetsInTemplateMixin ], {
    // Path to the template
    templateString : dojo.cache("loan", "templates/LoanInput.html"),
    principal: 0,
    interestPaid: 0,
    monthlyPayment: 0,

    // Override this method to perform custom behavior during dijit construction.
    // Common operations for constructor:
    // 1) Initialize non-primitive types (i.e. objects and arrays)
    // 2) Add additional properties needed by succeeding lifecycle methods
    constructor : function() {

    },

    // When this method is called, all variables inherited from superclasses are 'mixed in'.
    // Common operations for postMixInProperties
    // 1) Modify or assign values for widget property variables defined in the template HTML file
    postMixInProperties : function() {
    },

    // postCreate() is called after buildRendering(). This is useful to override when
    // you need to access and/or manipulate DOM nodes included with your widget.
    // DOM nodes and widgets with the dojoAttachPoint attribute specified can now be directly
    // accessed as fields on "this".
    // Common operations for postCreate
    // 1) Access and manipulate DOM nodes created in buildRendering()
    // 2) Add new DOM nodes or widgets
    postCreate : function() {
    },

    //this function will perform the calculations for our loan payment
    calculate: function() {
        this.principal = this.amount.attr('value');
    }
}

```

```
if(this.principal == NaN) {
  this.monthlyPayment = 0;
  this.principal = 0;
  this.interestPaid = 0;
} else {
  var interestRate = this.rate.attr('value') / 1200;
  var termInMonths = this.term.attr('value') * 12;

  this.monthlyPayment = Math.pow(1 + interestRate, termInMonths) - 1;
  this.monthlyPayment = interestRate + (interestRate / this.monthlyPayment);
  this.monthlyPayment = this.monthlyPayment * this.principal;

  this.interestPaid = (this.monthlyPayment * termInMonths) - this.principal;
}
}
});
});
```

Lesson checkpoint

You created a custom Dojo widget.

You learned:

- How to use content assist and templates to rapidly write Dojo code
- How to modify the HTML template for a custom Dojo widget
- How to modify the JavaScript file for a custom Dojo widget

[< Previous](#) | [Next >](#)

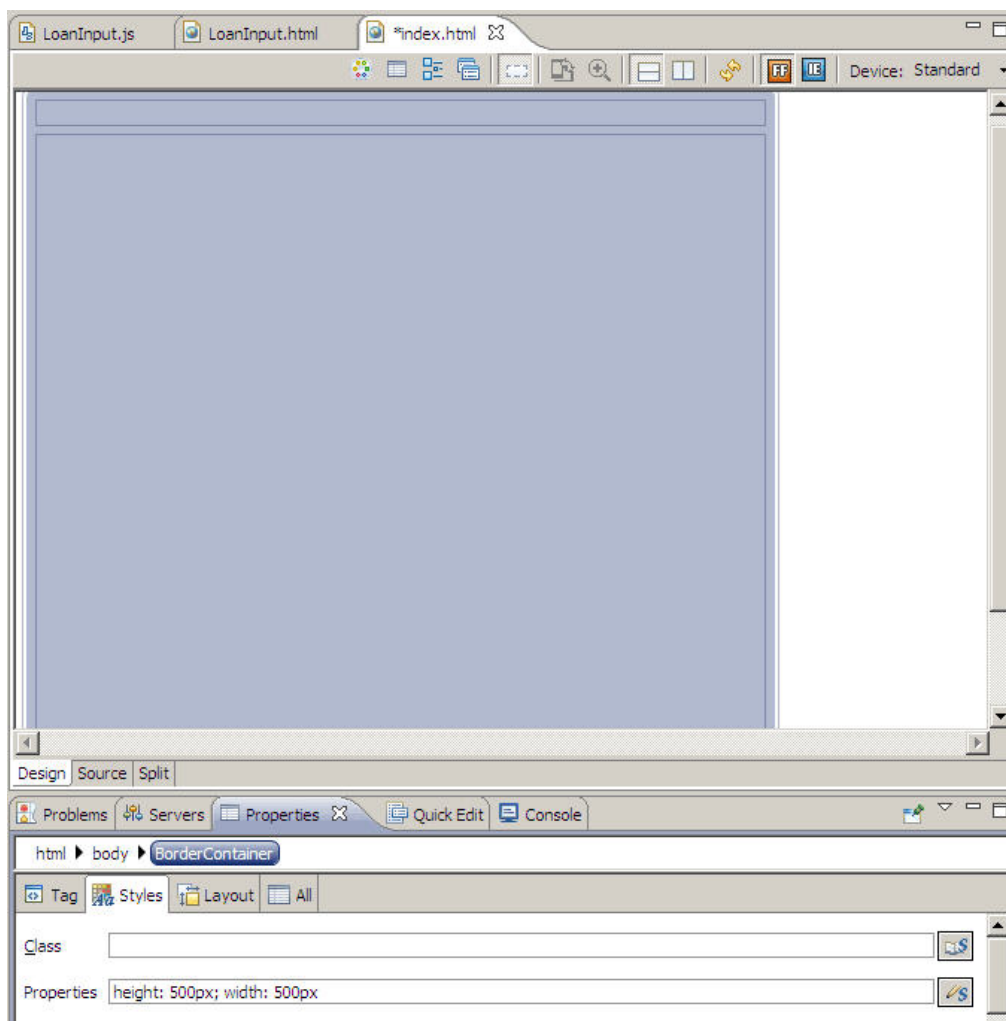
[< Previous](#) | [Next >](#)

Lesson 3: Add the custom Dojo widget to a web page

In this lesson you insert your custom Dojo widget into a web page that is laid out using a Dojo layout widget. You use Dojo APIs to connect your widget to the output field and display the results.

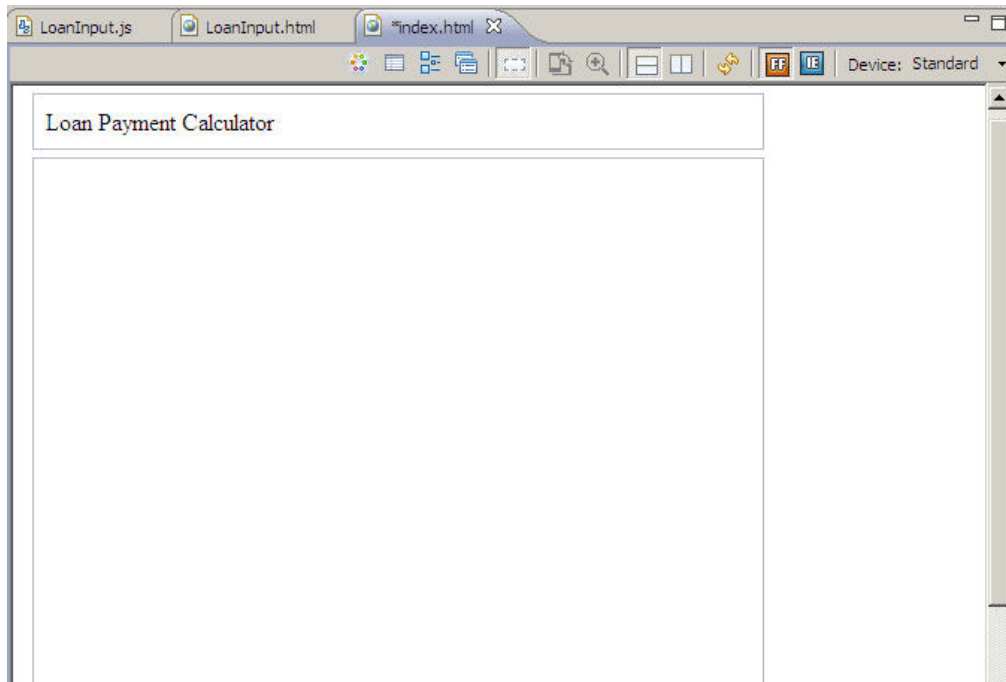
After you create a custom Dojo widget, the widget is added to the Other Dojo Widgets drawer of the Palette, making it easy to add the widget to the web page.

1. Right-click the `WebContent` folder and select **New > Web Page**.
2. Name the page `index.html` and click **Finish**.
3. Click the **Design** tab to display the page in the Design view.
4. In the palette, open the **Dojo Layout Widgets** drawer and drop a **BorderContainer** onto the page. The Insert Border Container dialog opens to allow you to customize the `BorderContainer` widget.
 - A. Select the **Top** and **Center** check boxes.
 - B. Click **OK**.
5. A visual view of the `BorderContainer` widget is now displayed in the Design view. Click the `BorderContainer` visualization and then click the **Properties** tab to open the Properties view.



6. Click the **Styles** tab and in the **Properties** field, update the following values for the border container:
 - Change the height from 500px to 700px.
 - Change the width to 600px.

7. Double-click the region to open a text box. Type `Loan Payment Calculator` and then click outside the text box to apply your change.



8. In the palette, expand the **Other Dojo Widgets** drawer.
9. Drop the **LoanInput** Dojo widget into the center region of the border container.
10. Click the **Source** tab to switch to the Source view.
11. In the `LoanInput` widget `div` tag, set the `id` attribute to `"LoanInput"`:

```
<div id="LoanInput" data-dojo-type="loan.LoanInput"></div>
```
12. Add a new `div` tag beneath the `LoanInput` widget to show the results. You can copy the following text:

```
<div>Monthly Payment: <span id="monthlyPayment"></span></div>
```
13. Add a new module `require` for `dijit/registry` immediately after the existing `dojo` `require`.

```
[ "dojo", "dijit/registry", "dojo/parser", "dijit/layout/BorderContainer", "dijit/layout/ContentPane", "loan/LoanInput" ],
```
14. Add a `registry` parameter to the `require` callback function.

```
function(dojoo, registry) {
  dojoo.ready(function() {
    });
  });
}
```
15. Inside the `dojoo.ready` function, perform the following steps:
 - A. Type `var loanWidget = registry.b` and invoke content assist. Select the **byId(id)** suggestion so that it inserts into your page.
 - B. Type `"LoanInput"` as the parameter for the `byId` function.
 - C. Type a semicolon after the closing parenthesis for the `LoanInput` parameter.
 - D. On the line under your call to `registry.byId`, type `dojoo.c` and invoke content assist. Here you can select another default template for `dojoo.connect`. There are two versions of this template. Choose the version that uses `dijit.registry.byId` and insert it into your page.
 - E. Set the first parameter as `LoanInput` and the second as `calculate`.
 - F. Inside the function parameter of the `connect` function, add the following code:

```
var payment = loanWidget.monthlyPayment;
if (payment == NaN) {
  payment = 0;
}
```

```

// update the result field
var formattedValue = dojo.currency.format(payment, {currency: "USD"});
dojo.attr("monthlyPayment", "innerHTML", formattedValue);

```

G. Required modules are added to the `require` function as an array of strings, where each module is a slash separated string of module segments. Inside the `require` function, add `"dojo/currency"` to the `require` dependencies array.

H. Your final code for the `dojo.ready` function should look like this:

```

dojo.ready(function() {
    // get the LoanInput widget
    loanWidget = registry.byId('LoanInput');

    // handle "calculate" from widget "LoanInput"
    dojo.connect(dijit.registry.byId("LoanInput"), "calculate", function(event) {
        var payment = loanWidget.monthlyPayment;
        if (payment == NaN) {
            payment = 0;
        }

        // update the result field
        var formattedValue = dojo.currency.format(payment, {currency: "USD"});
        dojo.attr("monthlyPayment", "innerHTML", formattedValue);
    });
});

```

16. Save the page. The source for the `index.html` file looks like the following:

```

<html>
<head>
<link rel="stylesheet" type="text/css"
href="dojo/dijit/themes/dijit.css">
<link rel="stylesheet" type="text/css"
href="dojo/dijit/themes/claro/claro.css">
<title>t1</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javascript"
data-dojo-config="isDebug: false, async: true, parseOnLoad: true"
src="dojo/dojo/dojo.js"></script>
<script type="text/javascript">
require(
// Set of module identifiers
[ "dojo", "dijit/registry", "dojo/parser", "dijit/layout/BorderContainer", "dijit/layout/ContentPane",
"loan/LoanInput", "dojo/currency" ],
// Callback function, invoked on dependencies evaluation results
function(doj, registry) {
    doj.ready(function() {
        // get the LoanInput widget
        loanWidget = registry.byId('LoanInput');

        // handle "calculate" from widget "LoanInput"
        doj.connect(dijit.registry.byId("LoanInput"), "calculate", function(event) {
            var payment = loanWidget.monthlyPayment;

```

```

    if (payment == NaN) {
        payment = 0;
    }

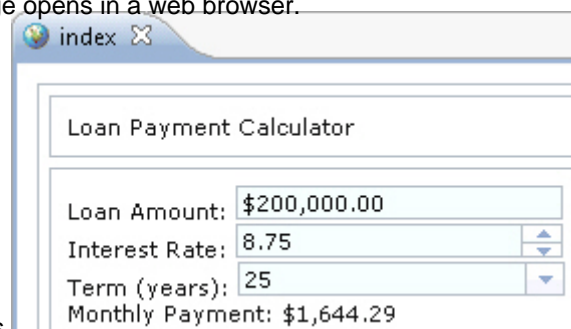
    // update the result field
    var formattedValue = dojo.currency.format(payment, {currency: "USD"});
    dojo.attr("monthlyPayment", "innerHTML", formattedValue);
    });
});
});
</script>
</head>
<body class="claro">
<div id="BorderContainer" style="height: 500px; width: 500px"
    data-dojo-type="dijit.layout.BorderContainer"
    data-dojo-props="design:'headline'">
<div data-dojo-type="dijit.layout.ContentPane"
    data-dojo-props="region:'top'">Loan Payment Calculator</div>
<div data-dojo-type="dijit.layout.ContentPane"
    data-dojo-props="region:'center'">
<div id="LoanInput" data-dojo-type="loan.LoanInput"></div>
<div>Monthly Payment: <span id="monthlyPayment"></span></div>
</div>
</div>

</body>
</html>

```

17. Now it is time to test your application on the server. In the Enterprise Explore view, right-click **index.html** and select **Run As > Run on Server**.

18. Select the **Web Preview Server** and click **Finish**. Your page opens in a web browser.



19. Enter a loan amount and verify that the results field updates.

Lesson checkpoint

You added your custom Dojo widget to the page and tested it on a server.

You learned:

- How to add a custom widget to a web page from the palette
- How to run a Dojo application on a server
- How to test the Dojo widgets that you created

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 4: Add a pie chart using the Dojo charting API (optional)

In this optional lesson, you use content assist and the Dojo charting API to add a pie chart to your results page. The pie chart displays the percentage of total loan costs going towards interest and principal.

Before you begin this lesson, [install Mozilla Firefox](#). **Note:** Steps may vary depending on the version of Firebug and Firefox you are using.

1. Configure the web application to run on Firefox by clicking **Window > Web Browser > Firefox**.
2. In the `index.html` file add the following `require` statements to the existing script tag:`"dojox/charting/Chart",`
`"dojox/charting/plot2d/Pie",`
`"dojox/charting/action2d/Highlight",`
`"dojox/charting/action2d/MoveSlice",`
`"dojox/charting/action2d/Tooltip",`
`"dojox/charting/themes/Dollar",`
3. Add the following div under the `MonthlyPayment` div you created to display the result:`<div id="chart" style="width: 350px; height: 350px"></div>`
4. In the existing `dojo.ready` function, type `var chart = new dojox` before the connect function and invoke content assist. You should see a list of all available Dojo types in the `dojox` namespace.
5. Continue typing `.charting.C` and you should see the list begin to filter down. Select **dojox.charting.Chart2D** and insert it on your page.
6. Add two parameters: `chart` and `null`, and the id for the div node where it will be located on your page:`var chart = new dojox.charting.Chart("chart", null);`
7. On the next line type `chart.set` and invoke content assist. You should see the `setTheme` method. Select this method and insert it into the page.
8. Enter `dojox.charting.themes.Dollar` as the parameter:`chart.setTheme(dojox.charting.themes.Dollar);`
9. Copy the following code on the next line. You can invoke content assist on the various methods and types if you want.

```
chart.addPlot("default", {
    type: "Pie",
    labelOffset: -30,
    radius: 90
});
chart.addSeries("paymentSeries", []);

new dojox.charting.action2d.MoveSlice(chart, "default");
new dojox.charting.action2d.Highlight(chart, "default");
new dojox.charting.action2d.Tooltip(chart, "default");
```

This code adds plotting information to your chart and sets up highlighting and tooltips for when users hover over the pie chart slices. For more information on Dojo charting APIs see: www.dojotoolkit.org/reference-guide/dojox/charting.html#dojox-charting.

10. Inside the `dojo.connect` function, under the existing code, add the following code:`// add the data series to the chart and render`

```
chart.updateSeries("paymentSeries", [
    {
        y: loanWidget.principal,
```

```

        stroke: "black",
        tooltip: "Principle"
    },
    {
        y: loanWidget.interestPaid,
        stroke: "black",
        tooltip: "Interest"
    }
]);
chart.render()

```

This code adds a new series of data to the chart and renders it each time that users change an input value. The resulting source is similar to the following:<!DOCTYPE HTML>

```

<html>
<head>
<link rel="stylesheet" type="text/css"
href="dojo/dijit/themes/dijit.css">
<link rel="stylesheet" type="text/css"
href="dojo/dijit/themes/claro/claro.css">
<title>t1</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javascript"
data-dojo-config="isDebug: false, async: true, parseOnLoad: true"
src="dojo/dojo/dojo.js"></script>
<script type="text/javascript">
require(
// Set of module identifiers
[ "dojo", "dijit/registry", "dojo/parser", "dijit/layout/BorderContainer", "dijit/layout/ContentPane",
"loan/LoanInput", "dojo/currency", "dojox/charting/Chart",
"dojox/charting/plot2d/Pie",
"dojox/charting/action2d/Highlight",
"dojox/charting/action2d/MoveSlice",
"dojox/charting/action2d/Tooltip",
"dojox/charting/themes/Dollar", ],
// Callback function, invoked on dependencies evaluation results
function(dojo, registry) {
dojo.ready(function() {
var chart = new dojox.charting.Chart("chart", null);
chart.setTheme(dojox.charting.themes.Dollar);
chart.addPlot("default", {
type : "Pie",
labelOffset : -30,
radius : 90
});
chart.addSeries("paymentSeries", []);

new dojox.charting.action2d.MoveSlice(chart, "default");
new dojox.charting.action2d.Highlight(chart, "default");
new dojox.charting.action2d.Tooltip(chart, "default");

```

```

// get the LoanInput widget
loanWidget = registry.byId('LoanInput');

// handle "calculate" from widget "LoanInput"
dojo.connect(dijit.registry.byId("LoanInput"), "calculate", function(event) {
    var payment = loanWidget.monthlyPayment;
    if (payment == NaN) {
        payment = 0;
    }

    // update the result field
    var formattedValue = dojo.currency.format(payment, {currency: "USD"});
    dojo.attr("monthlyPayment", "innerHTML", formattedValue);

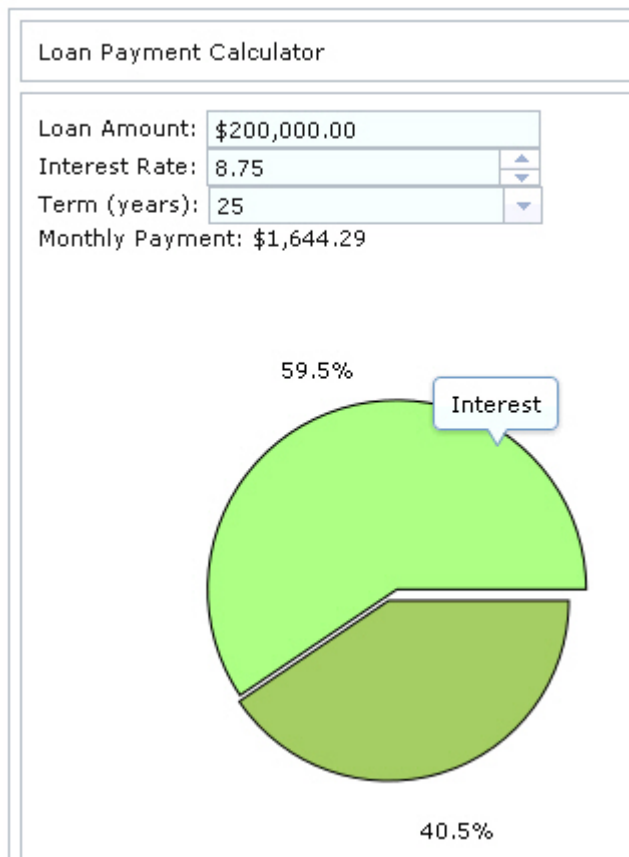
    // update the chart
    chart.updateSeries("paymentSeries", [ {
        y : loanWidget.principal,
        stroke : "black",
        tooltip : "Principal"
    }, {
        y : loanWidget.interestPaid,
        stroke : "black",
        tooltip : "Interest"
    } ]);
    chart.render();
});
});
});
</script>
</head>
<body class="claro">
<div id="BorderContainer" style="height: 500px; width: 500px"
    data-dojo-type="dijit.layout.BorderContainer"
    data-dojo-props="design:'headline'">
<div data-dojo-type="dijit.layout.ContentPane"
    data-dojo-props="region:'top'">Loan Payment Calculator</div>
<div data-dojo-type="dijit.layout.ContentPane"
    data-dojo-props="region:'center'">
<div id="LoanInput" data-dojo-type="loan.LoanInput"></div>
<div>Monthly Payment: <span id="monthlyPayment"></span></div>
<div id="chart" style="width: 350px; height: 350px"></div>
</div>
</div>

</body>
</html>

```

11. Save the page and run the page on server.

12. Enter a loan amount and view the pie chart.



Lesson checkpoint

You added a pie chart to your page and tested it on a server.

You learned:

- How to use content assist and the Dojo charting API to add a pie chart to a web page

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 5: Create a Dojo custom build (optional)

This optional lesson highlights the steps required to create a Dojo custom build. The purpose of a custom Dojo build is to create an efficient version of Dojo, and of your code, that is suitable for deployment.

Learn more about the Dojo Build System here: www.dojotoolkit.org/reference-guide/build/index.html#build-index

1. In the Enterprise Explorer view, right-click the **LoanPaymentCalculator** project and select **New > Dojo Custom Build**.
The Dojo Build Tools wizard opens.
2. Accept the default **Profile** location. Ensure that you complete a full build of the profile before you build individual layers.
3. Specify the build scripts and output directories. You can accept the default values.
4. Click **Override profile settings with command line** to specify an **Optimization** method. You can specify whether to delete output directories before building, copy test files into the build, or intern widget templates. When you intern a template, the HTML or CSS file is brought into the JavaScript file and assigned to a string.
5. To specify advanced options, click **Next**.
6. Click **Finish**. The Custom Build Output window opens displaying details of the build operation.

The Dojo custom build built the entire Dojo distribution and the Dojo layer files that you selected into the output folder that you specified in the Dojo Build Utility wizard.

Lesson checkpoint

You ran a custom build on your Dojo project.

You learned:

- How to optimize your Dojo code for deployment.

[< Previous](#) | [Next >](#)

< [Previous](#) | [Next](#) >

Lesson 6: Debugging your Dojo application with Firebug (optional)

This optional lesson highlights how to use Firebug to debug your web application.

Before you begin this lesson, [install Mozilla Firefox](#). **Note:** Steps may vary depending on the version of Firebug and Firefox you are using.

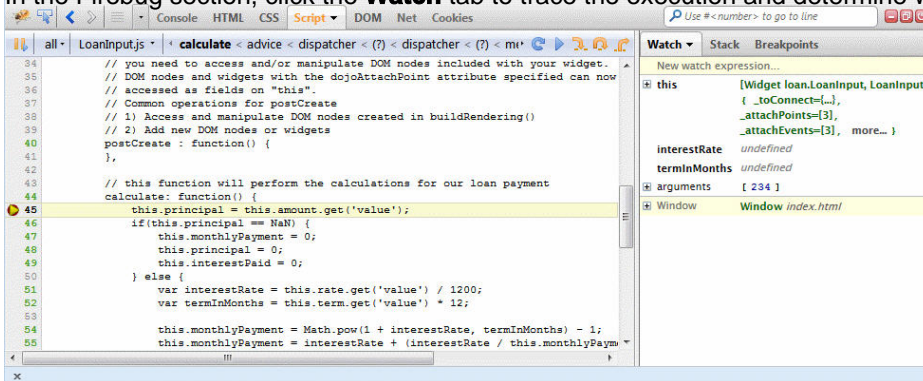
1. To configure the web application to run on Firefox, select **Window > Web Browser > Firefox**.
2. In the Enterprise Explorer view, right-click `index.html` and select **Debug As > Debug on Server**.
3. Click **Finish**. Your new web page opens in Firefox.
4. To enable Firebug, click  in the browser window frame.
5. In the Firebug section, click the **Script** tab and then click **Enable**.
6. After the scripts are enabled, click **Reload**. The web application is ready for debugging.
7. In the `LoanInput.js` file, select your custom Dojo widget JavaScript resource and then set a breakpoint.

```

31     },
32
33     // postCreate() is called after buildRendering(). This is useful to override when
34     // you need to access and/or manipulate DOM nodes included with your widget.
35     // DOM nodes and widgets with the dojoAttachPoint attribute specified can now be directly
36     // accessed as fields on "this".
37     // Common operations for postCreate
38     // 1) Access and manipulate DOM nodes created in buildRendering()
39     // 2) Add new DOM nodes or widgets
40     postCreate : function() {
41     },
42
43     // this function will perform the calculations for our loan payment
44     calculate: function() {
45         this.principal = this.amount.get('value');
46         if(this.principal == NaN) {
47             this.monthlyPayment = 0;
48             this.principal = 0;
49             this.interestPaid = 0;
50         } else {
51             var interestRate = this.rate.get('value') / 1200;
52             var termInMonths = this.term.get('value') * 12;
53
54             this.monthlyPayment = Math.pow(1 + interestRate, termInMonths) - 1;
55             this.monthlyPayment = interestRate + (interestRate / this.monthlyPayment);
56             this.monthlyPayment = this.monthlyPayment * this.principal;
57
58             this.interestPaid = (this.monthlyPayment * termInMonths) - this.principal;
59
60             //generate the Amortization Data
61             this.generateAmortizationSchedule(this.principal, interestRate, termInMonths, this.monthlyPayment);
62         }
63     },

```

8. In your browser, enter a loan amount and select an interest rate. The breakpoint that is set on line 45 stops the execution.
9. In the Firebug section, click the **Watch** tab to trace the execution and determine which values to change.



Tip: You can enable, disable, or

remove breakpoints from the **Breakpoints** tab.

10. Continue to use the Firebug debugging tools to evaluate your code. For more information about debugging JavaScript with Firebug, see getfirebug.com/wiki/index.php/Main_Page.

Lesson checkpoint

You debugged your Dojo files with Firebug.

You learned:

- How to open Firebug
- How to set breakpoints

Download the solution

Note: You can also download the completed web project and then run the project on the server to see the results of this tutorial.

Related information:

[🔗 FTP download for the sample](#)

[🔗 HTTP download for the sample](#)

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Summary: Create a loan payment calculator with Dojo

You learned how to create a Dojo-based loan payment calculator.

Lessons learned

By completing this tutorial you learned about the following concepts and tasks:

- How to create a Dojo-enabled web application to contain all of the resources required by your application.
- How to create a custom Dojo widget.
- How to write JavaScript and HTML code to create the Dojo user interface for your application.
- How to use content assistance, templates and wizards to write Dojo code and HTML.
- How to deploy your project to a server.
- How to create a Dojo custom build.
- How to debug your web application using Firebug.
- About the Dojo Toolkit.

Additional resources

For more information about the Dojo toolkit, see dojotoolkit.org. Documentation for the Dojo widgets is available at docs.dojocampus.org/.

[< Previous](#) | [Next >](#)

Web services tutorials

The Web services tutorials provide step by step instructions for you to follow, to complete specific tasks relating to Web service applications.

The Web services tutorials are contained in the following subsections:

- **Tutorial: Creating web services and an EJB skeleton from a WSDL file**

- **Tutorial: Creating a JAX-RS web service**

The following tutorial will walk you through creating a JAX-RS application. Any JAX-RS v1.0 implementation can be used, but for this tutorial we will be using the IBM® implementation that is available when you install WebSphere® Application Server Version 7.0 or later.

- **Tutorial: Creating a secured JAX-WS web service from an WSDL file**

This tutorial will walk you through the steps to create a JAX-WS web service and client, and to secure it using a policy set. It will generate code similar to that in the JAX-WS RSP address book sample.

Parent topic: [Tutorials](#)

Tutorial: Creating web services and an EJB skeleton from a WSDL file

Learning objectives

Important: Applicable edition: Full profile

You will learn how to create a WS-I compliant JAX-WS web service from a WSDL file. The wizard generates an EJB skeleton that contains a set of methods corresponding to the operations described in the WSDL document. When the EJB is created, each method has a trivial implementation that can be easily replaced by editing the EJB.

Time required

To complete this tutorial, you will need approximately **1 hour and 30 minutes**. If you decide to explore other facets of web services or EJBs while working on the tutorial, it could take longer to finish.

Prerequisites

In order to complete this tutorial end to end, you should be familiar with:

- Basic web services concepts, such as SOAP and WSDL
- Basic XML
- EJB programming


When you are ready, begin [Lesson 1.1: Set up the workspace and create the required projects](#)

[< Previous](#) | [Next >](#)

Lesson 1.1: Set up the workspace and create the required projects

Create a WebSphere Application Server

To create a WebSphere® Application Server, do the following:

1. From the **File** menu, select **New > Other > Server > Server > Next**.
2. Select the appropriate version of WebSphere Application Server as the server type. Click **Next**.
3. If this runtime has not been created in your workspace, you will be prompted to select the installation directory for the server. Click **Next**.
4. Accept the default server port and name. For this tutorial the default server name used will be `server1`. Click **Finish**.
5. Wait for the server to start. After it has started the Console view will display `Server server1 open for e-business`. If the server does not start automatically select it in the Servers view and click the start icon: .

Setting the WS-I compliance level

WS-I refers to web service interoperability; this includes interoperability across platforms, operating systems, and programming languages.

The WS-I organization sets out standards collected in documents called Profiles that define the requirements needed to make a web service interoperable. The Rational® Developer products validate web services against the WS-I Simple SOAP Binding Profile 1.0 (WS-I SSBP) and the WS-I Attachments Profile 1.0 (WS-I AP). For more information on WS-I, refer to their Web site: <http://www.ws-i.org/>

By default, the WS-I SSBP compliance level is set to **Ignore**. With this setting, no warning will be given if non-compliant choices are made. This compliance level is used by the web service wizards and the WSDL validation tool. This sample generates a WS-I compliant web service, therefore you should set the WS-I compliance level to **Require**.

You can change the WS-I compliance level by following the proceeding steps:

1. On the main menu bar, click **Window > Preferences**. The Preferences dialog box opens.
2. Expand the **General > Service Policies** branch and expand **Profile Compliance > WS-I BP 1.1 + SSBP 1.0**, and select the **Require compliance** option from the drop-down list
3. Click **OK**.

Creating the web service EJB project

The remaining steps in this tutorial will be done in the Java™ EE perspective. If you are asked if you want to change to another perspective after performing a task, select **No**.

The EJB project will contain the business logic for the web service as well as the WSDL file.

1. On the main menu bar, click **File > New > Project > EJB > EJB Project**. Click **Next**.
2. Type `TempEJB` in the Project name text field. Under Target Runtime ensure that the target server is the appropriate version of WebSphere Application Server. In the **EAR Project Name** field, enter `TempEJB_EAR` as the EAR name. Click **Next**.

3. Clear the checkbox for creating a client JAR module. The web services wizard will create this module for you. Click **Finish**.

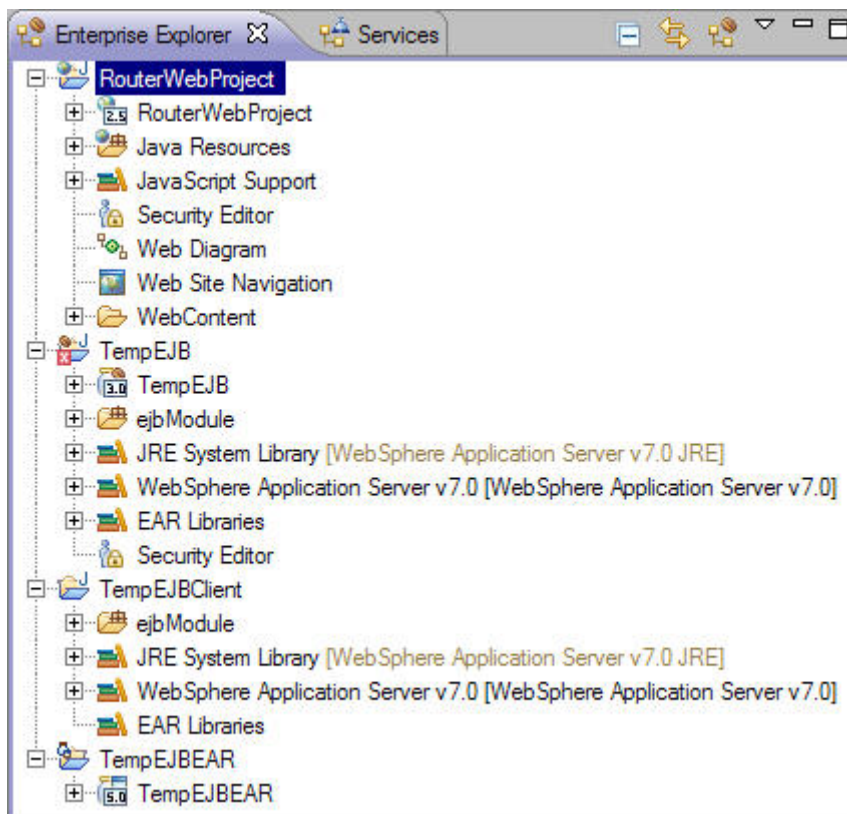
The EJB project that will contain the web service logic and the associated EAR are created. The EJB project will have an error associated with it because it does not contain an enterprise bean. The bean will be generated by the web services wizard.

Creating the web service router project

EJB web services require a router project. This project contains the router servlet that acts as the endpoint for the service and will call out to the EJB. If you are using SOAP over JMS as your transport method the router project needs to be an EJB project. If you are using SOAP over HTTP as we are in this tutorial, the router project should be a Web project. The project you create must be added to the same EAR as the EJB project that will contain the enterprise bean. This project should not contain any of the business logic for your web service.

You can create a Web project by following these steps:

1. On the main menu bar, click **File > New > Project > Web > web Project**. Click **Next**.
2. Type `RouterWebProject` in the Name text field. Under Target Runtime ensure that the target server is the appropriate WebSphere Application Server. In the **EAR Project Name** field, ensure `TempEJBEAR` is selected. This will ensure that the enterprise bean that you will create later and your router project are both referenced in the same EAR. Click **Finish**.
3. You have now created your router project and your workspace should look similar to the following:



Adding the projects to the server

You can associate the project with the server that your web service will run on by following these steps:

1. Right-click the server in the Servers view and select **Add and Remove**. If the Servers view is not open in your workspace, open it from the **Window** menu by selecting **Show View > Servers**.

2. In the window that opens, select TempEJBEAR which contains your router and EJB projects, and click **Add**.
3. Click **Finish**.

Lesson checkpoint

Now you are ready to begin [Lesson 1.2: Import and validate the WSDL file](#).

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 1.2: Download and validate the WSDL file

The Temperature Conversion WSDL document has been provided for you. The WSDL file that you will use in this tutorial converts temperature from Fahrenheit to Celsius and Celsius to Fahrenheit.

Before you begin, you must complete [Lesson 1.1: Set up the workspace and create the required projects](#).

You can create a new WSDL document or download an existing one. The Temperature Conversion WSDL document used in this tutorial has been provided for you in a simple project. Complete the following steps to download the Temperature Conversion WSDL document into the workbench:

1. In the Enterprise Explorer view, expand **TempEJB**, right-click and click **New > Folder** to create a folder called `wsdl`.
Click **Finish**.
2. Download the simple project containing the WSDL file
3. Expand the downloaded simple project called TempConversionWSDL. The ConvertTemperature.wsdl file will be located here.
4. Right-click the ConvertTemperature.wsdl file and select **Move**.
5. Select the `wsdl` folder you created in the TempEJB project as your destination and click **OK**.

Validating the WSDL document

The WSDL Validator can validate WSDL semantics and WS-I compliance.

You can validate the Temperature Converter WSDL document by following the proceeding steps:

1. Select the ConvertTemperature.wsdl document in the Project Navigator.
2. Right-click, and click **Validate**. Any errors will be displayed in the Tasks view.

If no errors occur during validation, you can proceed to create the web service.

Lesson checkpoint


Now you are ready to begin [Lesson 1.3: Create the web service](#).

[< Previous](#) | [Next >](#)

Lesson 1.3: Create the web service

Before you begin, you must complete [Lesson 1.2: Import and validate the WSDL file](#).

Before you attempt to create a web service it is strongly suggested that you start the WebSphere® Application Server on which the web service will run. Although you can start the server in the web service wizards, since it may take several minutes to start depending on the speed of your machine, starting the server before you begin will both increase the speed with which you complete the wizard and reduce the chance that the wizard will generate an error because the server is taking too long to start.

To start the server, select the server in the Servers view and select **Start:** 

If the Servers view is not open in your workspace, open it from the **Window** menu by selecting **Show View > Servers**.

Create a web service from a WSDL file

The web service wizard assists you in creating a new web service, configuring it for deployment, and deploying the web service to a server. Once your web service is deployed, the wizard assists you in generating the client proxy and sample application to test the web service.

1. In the Project Explorer, select the **ConvertTemperature.wsdl** document in your EJB project.
2. Click **File > New > Other**. Select **Web Services** in order to display the various web service wizards. Select the **Web Service** wizard. Click **Next**.
3. Select the following options on the first page of the wizard:
 - Web service type: Top down EJB Web service
 - Service definition: Ensure the ConvertTemperature.wsdl file that you imported is selected.
 - Level of service generation slider: move the slider to Test service. The slider sets the defaults on the remaining wizard pages, but you can override the default settings on each page as you proceed.
 - Service configuration: Ensure that WebSphere Application Server Version 7 or later and the IBM® WebSphere JAX-WS runtime environment are selected. Click **Service project** and enter `TempEJB` as your service project name. `TempEJB` should be selected as your service EAR project.
 - Level of client generation slider: move the slider to Test client.
 - Client configuration: Ensure that WebSphere Application Server Version 7 or later and the IBM WebSphere JAX-WS runtime are selected. The wizard will create a client and client EAR project. You can accept the default names or enter a different name.
 - Monitor the web service.Click **Next**.
4. On the Web Service Configuration page, leave all the default options selected, and click **Next**.
5. On the Router project configuration page, select `RouterWebProject` as your http router project if it is not already selected, and click **Next**.
6. In the Web Service Test page, you can select a test facility to test your web service before a client or proxy is developed. Select Web Services Explorer as the test facility for your web service and click **Launch**. This step may take several seconds for the WebSphere Application server to start.
7. The Web Services Explorer is displayed in a Web browser. Select **fahrenheitToCelsius** or **celsiusToFahrenheit** from the operations list. Enter a number in the value field and click **Go**. A trivial implementation of each of these operations is provided, and a default value of -3 is returned. If both operations complete successfully, close the browser window and click **Next** in the web services wizard.

8. In the Web Service Client Configuration page, keep the default selections. Click **Next**.
9. In the Web Service Client Test page, ensure **Test the generated proxy** and **Run test on server** are both selected. In the Methods section. Ensure that all methods are selected, or click **Select All** to select all methods. Click **Finish**.
10. The sample application is launched in a Web browser. You can use this application to test the web service by selecting a method in the Methods frame, entering an input value in the Inputs frame, and clicking **Invoke** to view the result in the Result frame. Do not close the TestClient.jsp browser window yet - it will be used to test the web service traffic for WS-I compliance later in this tutorial.

Lesson checkpoint

Now you are ready to begin [Lesson 1.4: Implement the temperature conversion methods](#).

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 1.4: Implement the temperature conversion methods

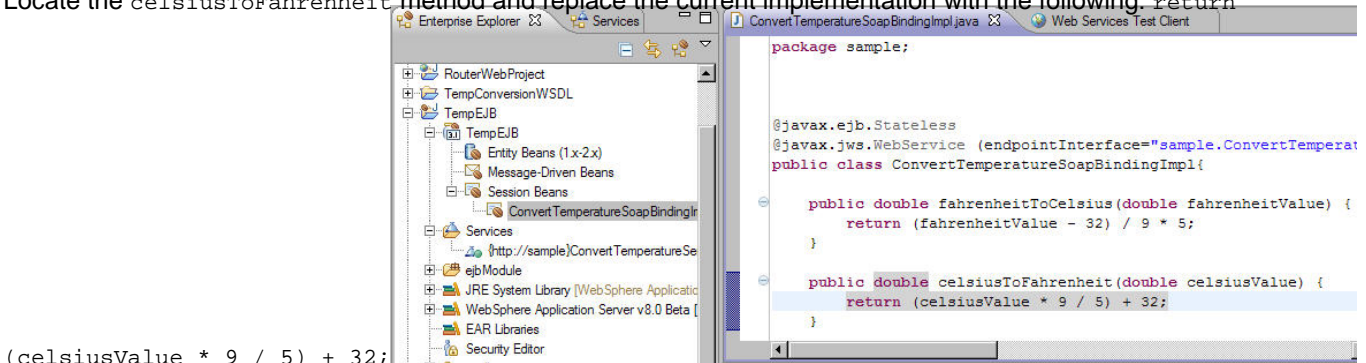
Before you begin, you must complete [Lesson 1.3: Create the web service](#).

Trivial implementations of the **fahrenheitToCelsius** and **celsiusToFahrenheit** methods were automatically generated when you created a web service from your WSDL document. In this section you will replace these trivial implementations with more meaningful code, and perform the necessary steps to test your new methods.

1. In the Enterprise Explorer view, select **ConvertTemperatureSoapBindingImpl.java** under **TempEJB > Session Beans**

2. Locate the `fahrenheitToCelsius` method and replace the current implementation with the following: `return (fahrenheitValue - 32) / 9 * 5;`

3. Locate the `celsiusToFahrenheit` method and replace the current implementation with the following: `return`



`(celsiusValue * 9 / 5) + 32;`

4. Save your updates by clicking **File > Save**.

5. Restart the EAR by expanding your server in the Servers view and right-clicking **TempEJBEAR > Restart TempEJBEAR**.

6. In the Web Services Test Client, test your **fahrenheitToCelsius** and **celsiusToFahrenheit** methods.

Lesson checkpoint

Now you are ready to begin [Lesson 1.5: Validate the web service traffic WS-I compliance](#).

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 1.5: Validate the web service traffic WS-I compliance

Before you begin, you must complete [Lesson 1.4: Implement the temperature conversion methods](#).

To ensure that the SOAP envelope request and response pairs are WS-I compliant, you need to direct your web service traffic through the TCP/IP Monitor:

When creating a web service using the web service or web service client wizards, you can select to set up and run the TCP/IP Monitor automatically. Since you chose this option when creating the web service, the TCP/IP monitor view should be in your workspace. If it is not, you can open this view by selecting **Window > Show View > Other > Debug > TCP/IP Monitor**.

Alternately, you can set up the TCP/IP Monitor manually by completing the following steps:

1. In the sample application, invoke the getEndPoint method. Record this endpoint.
2. Create a server to act as the TCP/IP Monitor:
 - A. From the **Window** menu, select **Preferences**.
 - B. In the Preferences window, expand **Run/Debug** and then select **TCP/IP Monitor**.
 - C. Select the **Show TCP/IP Monitor View when there is activity** check box.
 - D. Under the TCP/IP Monitors lists, click **Add**. A New Monitor dialog box opens.
 - E. Specify the following settings:

Option	Description
Local monitoring port	Specify a unique port number on your local machine.
Host name	Specify the host name or IP address of the machine where the server is running.
Port	Specify the port number of the remote server.
Type	Specify whether the request type from the Web browser are sent by HTTP or TCP/IP. If the HTTP option is selected the requests from the Web browser are modified so that the HTTP header points to the remote machine and separated if multiple HTTP requests are received in the same connection. If the TCP/IP option is selected, all the requests are sent byte for byte.

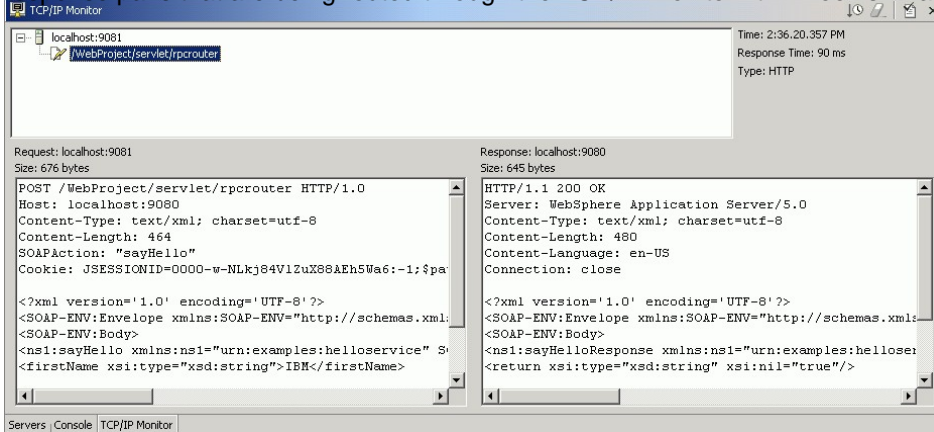
3. In order to route the web service through the monitor, the endpoint of the web service client needs to be changed. The TCP/IP Monitor listens on port 9081. In the Web browser window used in step 1, invoke the setEndPoint method, and change the endpoint so that it directs to port 9081. For example, the default would be:


`http://localhost:9081/web_module_context_root/servlet/rpcrouter` Invoke the getEndPoint method again to ensure that your change has been implemented.

Routing traffic and verifying WS-I compliance

You can route traffic through the TCP/IP monitor and test the traffic for WS-I compliance by following the proceeding steps:

1. Select a web service method in the Methods pane. Invoke this method.
2. Change to the TCP/IP Monitor view by clicking the TCP/IP Monitor tab in the Servers view. This will display request and response pairs that are being routed through the TCP/IP Monitor. It will look similar to the following picture:



3. To ensure that your web service SOAP traffic is WS-I compliant, you can generate a log file by clicking the  icon. In the dialog box that opens, select a name for the log file and specify where you want it to be stored. This log file will be validated for WS-I compliance. You can open the log file in an XML editor to examine its contents.

Lesson checkpoint

Finish your tutorial by reviewing the materials in the [Summary](#).

[< Previous](#) | [Next >](#)

[< Previous](#)

Summary

You have created a WS-I compliant Temperature Conversion Web service and an EJB skeleton from a WSDL file, and learned to implement some basic EJB methods.

Lessons learned

If you have completed all of the exercises, you should now be able to:

- Set the WS-I compliance level.
- Create a Web project called WebProject.
- Import the Temperature Conversion WSDL document.
- Validate the WSDL, and validate the WSDL file's WS-I compliance.
- Create the web service and skeleton EJB, and test the methods included in the web service using the Web Services Explorer.
- Implement the fahrenheitToCelsius and celsiusToFahrenheit methods (optional).
- Deploy the web service to the WebSphere® Application Server.
- Test the web service traffic for WS-I compliance.

More information

For more information about web services, WSDL, SOAP, and the WebSphere runtime environments, consult the online help: [Developing web service applications](#)

For more in-depth technical articles on web services, consult [DeveloperWorks](#)

[< Previous](#)

Tutorial: Creating a JAX-RS web service

The following tutorial will walk you through creating a JAX-RS application. Any JAX-RS v1.0 implementation can be used, but for this tutorial we will be using the IBM® implementation that is available when you install WebSphere® Application Server Version 7.0 or later.

Learning objectives

Important: Applicable editions: Liberty profile, full profile

In this tutorial, you will learn to:

- Integrate a JAX-RS implementation into the workbench, and target projects to use this implementation.
- Create a bottom-up web service which uses JAX-RS, using quick fixes to speed up the process.

Time required

This tutorial requires 30 minutes to complete.

Prerequisites

Ensure that you have one of the following prerequisites:

- WebSphere Application Server Version 7.0 with the Feature Pack for Web 2.0 and Mobile
- WebSphere Application Server Version 8.0
- WebSphere Application Server Version 8.5

If you do not want to complete the tutorial manually, you can download the sample through HTTP or FTP, and save the compressed file to a directory on your workstation. Note that this sample code is targeted to WebSphere Application Server v8 and cannot be deployed to WebSphere Application Server v7. You can then proceed to the test section of Lesson 2.

If you want to watch a JAX-RS web service being created, a demonstration is available at the following URL: [Creating and testing JAX-RS \(RESTful\) web services](#)

When you are ready, begin [Creating a server and web project](#).

Related information:

[FTP download for the sample](#)

[HTTP download for the sample](#)


[< Previous](#) | [Next >](#)

Lesson 1: Creating a server and web project

The web service needs to reside in a web project with the JAX-RS facet enabled.

Create a JAX-RS enabled server

Before creating the web service, you need to have a server which has Java 5.0 or later JVM support defined and started. By default a server is created for you when you install WebSphere Application Server. This server can be seen in the Servers view. However, if you want to create a new WebSphere Application Server do the following:

1. From the **File** menu, select **New > Other > Server > Server > Next**.
2. Select the appropriate version of WebSphere Application Server as the server type. Click **Next**.
3. If this runtime has not been created in your workspace, you will be prompted to select the installation directory for the server. Click **Next**.
4. Accept the default server port and name. For this tutorial the default server name used will be `server1`. Click **Finish**.
5. Wait for the server to start. After it has started the Console view will display `Server server1 open for e-business`. If the server does not start automatically select it in the Servers view and click the start icon: .

Create a JAX-RS enabled web project

The JAX-RS web service needs to reside in a project with the JAX-RS facet enabled.

1. In the Java EE perspective, right-click your enterprise application project and select **New > Web Project** to open the web project wizard.
2. In the **Name** field, type a name for your new web project. For this tutorial, use `JAXRS`.
3. In the Project Templates section, select the type of web template you want to use: For this tutorial, select `Simple`.

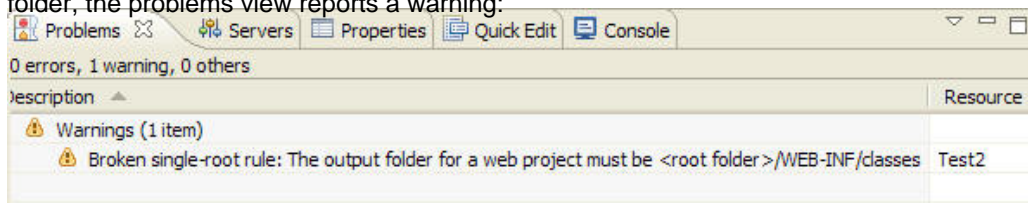
Option	Description
Dojo Toolkit	Configures the project to have Dojo capabilities. The Dojo resources can be in the project itself, a separate project, or a remote location accessible via HTTP.
JavaServer Faces	Enables the project to be deployed with JSF capabilities. Configuration is provided for either JSP or Facelets.
REST Services	A project configured for REST Services based on JAX-RS
Simple	This creates a basic web project.

4. In the Programming Model section, select the programming model that you want to use: For this tutorial, select `Java EE`.
 - Client-side only (HTML, JavaScript,...)
 - Java EE
 - OSGi
5. Click **Next** to configure your new web project.
6. On the deployment page, from the list of available configuration options, click **Deployment** to open the Deployment configuration page.
 - In the **Target runtime** field, select the WebSphere Application Server that you installed earlier in the tutorial.

- In the **Web module version** field, accept the default, which is automatically selected based on which WebSphere Application Server you selected.
- In the **EAR Membership** field, select **Add project to an EAR**, and ensure that `JAXRSEAR` is the EAR project name.
- Under the Deployment section, select **Change Features**. On the Project Facets page, select **JAX-RS (REST Web Services)**, version **1.1**, and click **OK**.

7. From the list of available configuration options, click **Java** to open the Java configuration page.

- In the **Source folders on build path** field, accept the default **src** directory, or click **Add Folder**, **Edit...** or **Remove** to specify a folder for your source files.
- In the **Default output folder:** field, specify a folder for your output files or accept the default value (`WebContent\WEB-INF\classes`). **Important:** If you choose a folder other than `WebContent\WEB-INF\classes` for your default output folder, the problems view reports a warning:



The default for single rootedness problems is set to warning. To change this setting, use the Validation Filters for Project Structure Validator page:

- Click **Window > Preferences > Validation > Project Structure Validation** and then the **...** button for **Settings**.
- In the Validation Filters for Project Structure Validator page, specify the default severity level. Available severity levels are **Error**, **Warning**, and **Ignore**.

8. From the list of available configuration options, click **REST Services** to open the REST Services configuration page. In the **JAX-RS Implementation Library** field, select **IBM WebSphere Application Server v<x> JAX-RS Library**.

Ensure that the following values appear:

- In the **JAX-RS servlet name:** field, ensure that **JAX-RS Servlet** appears.
- In the **JAX-RS servlet class name:** field, ensure that **com.ibm.websphere.jaxrs.server.IBMRestServlet** appears.
- In the **URL mapping patterns:** field, ensure that **/jaxrs/*** appears.

Learn more about libraries: The JAX-RS libraries for each WebSphere Application Server that you have installed will be listed in this drop-down box. If you want to use a non-WebSphere library, it can be imported using the User Libraries preference page. If you select User Library, you can launch the User Libraries preference page and add a library. After a project has been created, you can change the library on the **Preferences > Project Facets > JAX-RS** page.

9. From the list of available configuration options, click **Web Module**. On the Web Module configuration page:

- In the **Context root** field, type the name of your web project root, or accept the default (which is the name of your web project).
- In the **Content directory** field, type the name of your content directory, or accept the default (`WebContent`).
- Select **Generate web.xml deployment descriptor** if you want to create a deployment descriptor. You can also add a deployment descriptor to your web module later. You need to use a `web.xml` to configure security constraints and other behavior.

10. Click **Finish**.

The facet adds the library, servlet information, and support for JAX-RS annotations processing and JAX-RS quick-fixes.

Now you are ready to begin the next module: [Creating and testing the web service](#).

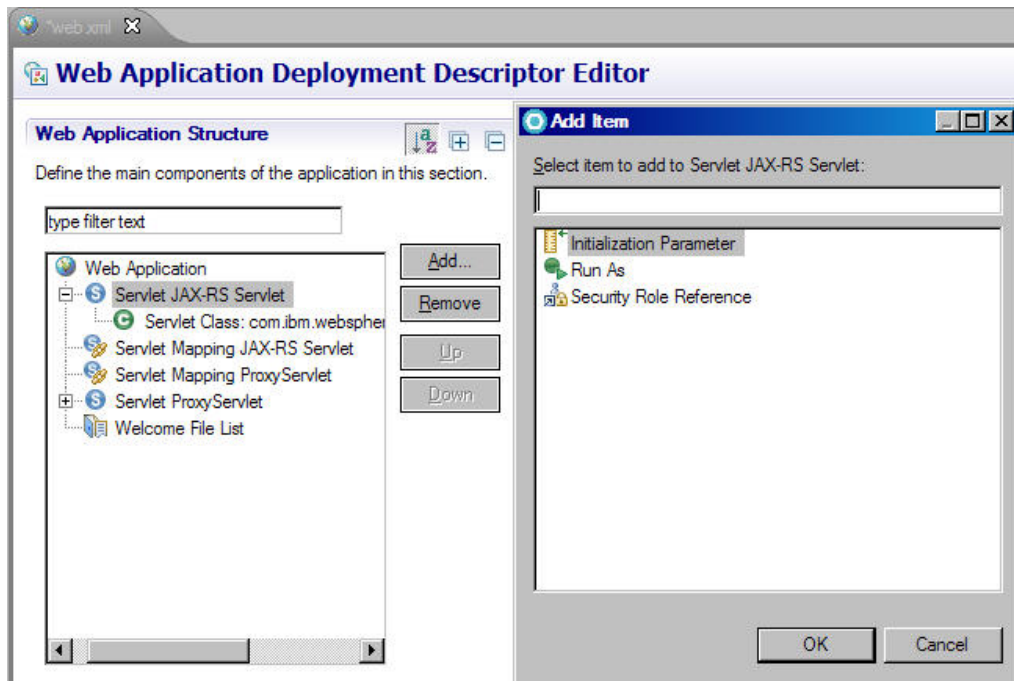
[< Previous](#)

Lesson 2: Creating and testing the web service

After you have created a JAX-RS enabled project, you can download the Java™ files used by the application and test the web service.

Create a JAX-RS web service

1. Download the project which contains the Java classes needed for the application.
2. In your web project, create a package called `com.test` (right-click **Java Resources** > **src** and select **New** > **Package**). Import the following classes from the downloaded project into the package:
 - `AddressBook.java`
 - `AddressBookApplication.java`
3. Open `WebContent/WEB-INF/web.xml`. In the Design view, select the Servlet (JAX-RS Servlet) and click **Add** and add an Initialization parameter to the JAX-RS servlet, leaving the name and value fields empty. Save `web.xml` ignoring any errors that might be displayed.



4. In the Problems view, right-click the `param-name` warning and select Quick Fix. Select to browse for an existing subclass, and select the `AddressBookApplication`.
5. Save `web.xml`.

Test the JAX-RS web service

1. In the Servers view, right-click your server and select **Add and Remove**, and add the JAX-RS EAR to the server. Restart the server.
2. To retrieve all addresses in the Address Book application, open a Web browser and enter the following URL:
`http://localhost:<default_host_port>/<application_name>/jaxrs/addresses` For example, following the naming convention used in this tutorial and the default port, `http://localhost:9080/JAXRS/jaxrs/addresses` **Note:** You can determine the default host port name in the WebSphere Application Server Admin Console server configuration tab.

3. Enter the following URL: `http://localhost:<default_host_port>/<application_name>/jaxrs/addresses/<address_index>` The `address_index` is a number between 0 and 5 which represent the 6 addresses listed in `AddressBook.java`. The address assigned to that index value will display.

Related information:

[FTP download for the sample](#)

[HTTP download for the sample](#)

[< Previous](#)

Tutorial: Creating a secured JAX-WS web service from an WSDL file

This tutorial will walk you through the steps to create a JAX-WS web service and client, and to secure it using a policy set.

It will generate code similar to that in the JAX-WS RSP address book sample.

Learning objectives

Important: Applicable edition: Full profile

In this tutorial, you will learn to:

- Create a JAX-WS enabled server and Web project
- Create the address book web service and test its business logic using the Generic Service Client
- Create the address book web service client and test it using sample JSPs
- Secure the web service using the WebSphere® Application Server

To complete this tutorial, you will need approximately 2 hours. If you decide to explore other facets of web services while working on the tutorial, it could take longer to finish.

Prerequisites

In order to complete this tutorial end to end, you should be familiar with:

- Web services concepts
- WebSphere Application Server concepts

When you are ready, begin [Lesson 1: Creating a server and Web project](#).


[< Previous](#) | [Next >](#)

Lesson 1: Creating a server and web project

In this lesson you will learn how to create a server and web project for use with web services.

Create a JAX-WS enabled server

Before creating the web service, you need to ensure you have a WebSphere® Application Server v8.0 server and up defined and started. By default a server is created for you when you install WebSphere Application Server. This server can be seen in the Servers view. However, if you want to create a new server do the following:

1. From the **File** menu, select **New > Other > Server > Server > Next**.
2. Select **WebSphere Application Server v8.0** as the server type. Click **Next**.
3. If this runtime has not been created in your workspace, you will be prompted to select the installation directory for the server. Click **Next**.
4. Accept the default server port and name. For this tutorial the default server name used will be `server1`. Click **Finish**.
5. Wait for the server to start. After it starts, the Console view will display `Server server1 open for e-business`. If the server does not start automatically select it in the Servers view and click the start icon: .

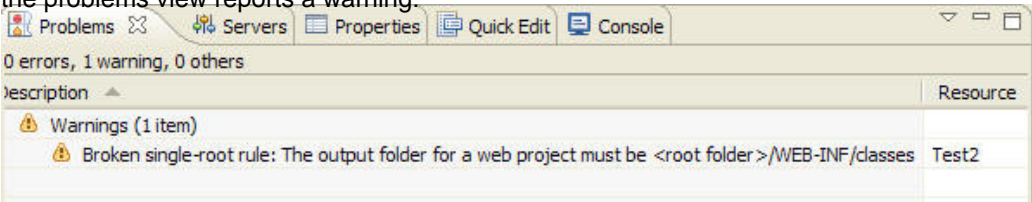
Create a web project for the web service

The web services wizards can create a web project for the web service and enable the facets for you, however in this tutorial you will create the project manually.

1. In the Java EE perspective, right-click your enterprise application project and select **New > Web Project** to open the web project wizard.
2. In the **Name** field, type a name for your new web project. For this tutorial, use `jwsAddressBook`.
3. In the Project Templates section, select the type of web template you want to use: For this tutorial, select `Simple`.

Option	Description
Dojo Toolkit	Configures the project to have Dojo capabilities. The Dojo resources can be in the project itself, a separate project, or a remote location accessible via HTTP.
JavaServer Faces	Enables the project to be deployed with JSF capabilities. Configuration is provided for either JSP or Facelets.
REST Services	A project configured for REST Services based on JAX-RS
Simple	This creates a basic web project.

4. In the Programming Model section, select the programming model that you want to use: For this tutorial, select `Java EE`.
 - Client-side only (HTML, JavaScript,...)
 - Java EE
 - OSGi
5. Click **Next** to configure your new web project.
6. On the deployment page, from the list of available configuration options, click **Deployment** to open the Deployment configuration page.
 - In the **Target runtime** field, select the v7 or v8 WebSphere Application Server that you installed earlier in the tutorial.

- In the **Web module version** field, accept the default (which is automatically selected based on which WebSphere Application Server you selected).
 - In the **EAR Membership** field, select **Add project to an EAR**, and ensure that `jwsAddressBookEAR` is the EAR project name.
 - Under the Deployment section, select **Change Features**. On the Project Facets page, select **JAX-RS (REST Web Services)**, version **1.1**, and click **OK**.
7. From the list of available configuration options, click **Java** to open the Java configuration page.
- In the **Source folders on build path** field, accept the default **src** directory, or click **Add Folder**, **Edit...** or **Remove** to specify a folder for your source files.
 - In the **Default output folder:** field, specify a folder for your output files or accept the default value (`WebContent\WEB-INF\classes`). **Important:** If you choose a folder other than `WebContent\WEB-INF\classes` for your default output folder, the problems view reports a warning:
- 
- The default for single rootedness problems is set to warning. To change this setting, use the Validation Filters for Project Structure Validator page:
- A. Click **Window > Preferences > Validation > Project Structure Validation** and then the **...** button for **Settings**.
 - B. In the Validation Filters for Project Structure Validator page, specify the default severity level. Available severity levels are **Error**, **Warning**, and **Ignore**.
8. From the list of available configuration options, click **Web Module**. On the Web Module configuration page:
- In the **Context root** field, type the name of your web project root, or accept the default (which is the name of your web project).
 - In the **Content directory** field, type the name of your content directory, or accept the default (`WebContent`).
 - Select **Generate web.xml deployment descriptor** if you want to create a deployment descriptor. You can also add a deployment descriptor to your web module later. You need to use a `web.xml` to configure security constraints and other behavior.
9. Click **Finish**.

Create a web project for the web service client

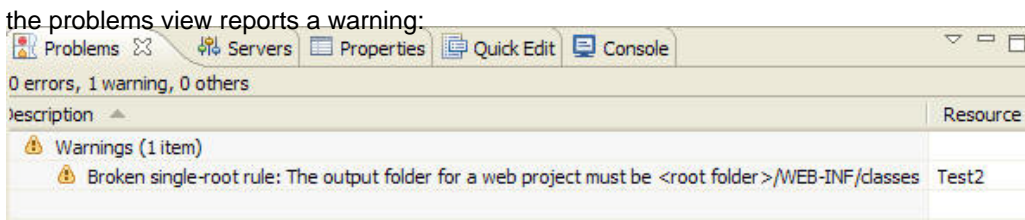
The web services wizards can create a web project for the client and enable the facets for you, however in this tutorial you will create the project manually.

1. In the Java EE perspective, right-click your enterprise application project and select **New > Web Project** to open the web project wizard.
2. In the **Name** field, type a name for your new web project. For this tutorial, use `jwsAddressBookClient`.
3. In the Project Templates section, select the type of web template you want to use: For this tutorial, select `Simple`.

Option	Description
Dojo Toolkit	Configures the project to have Dojo capabilities. The Dojo resources can be in the project itself, a separate project, or a remote location accessible via HTTP.

JavaServer Faces	Enables the project to be deployed with JSF capabilities. Configuration is provided for either JSP or Facelets.
REST Services	A project configured for REST Services based on JAX-RS
Simple	This creates a basic web project.

4. In the Programming Model section, select the programming model that you want to use: For this tutorial, select `Java EE`.
 - Client-side only (HTML, JavaScript,...)
 - Java EE
 - OSGi
5. Click **Next** to configure your new web project.
6. On the deployment page, from the list of available configuration options, click **Deployment** to open the Deployment configuration page.
 - In the **Target runtime** field, select the v7 or v8 WebSphere Application Server that you installed earlier in the tutorial.
 - In the **Web module version** field, accept the default (which is automatically selected based on which WebSphere Application Server you selected).
 - In the **EAR Membership** field, select **Add project to an EAR**, and ensure that `jwsAddressBookEAR` is the EAR project name.
 - Under the Deployment section, select **Change Features**. On the Project Facets page, select **JAX-RS (REST Web Services)**, version **1.1**, and click **OK**.
7. From the list of available configuration options, click **Java** to open the Java configuration page.
 - In the **Source folders on build path** field, accept the default `src` directory, or click **Add Folder**, **Edit...** or **Remove** to specify a folder for your source files.
 - In the **Default output folder:** field, specify a folder for your output files or accept the default value (`WebContent\WEB-INF\classes`). **Important:** If you choose a folder other than `WebContent\WEB-INF\classes` for your default output folder, the problems view reports a warning:



The default for single rootedness problems is set to warning. To change this setting, use the Validation Filters for Project Structure Validator page:

- A. Click **Window > Preferences > Validation > Project Structure Validation** and then the **...** button for **Settings**.
- B. In the Validation Filters for Project Structure Validator page, specify the default severity level. Available severity levels are **Error**, **Warning**, and **Ignore**.

8. From the list of available configuration options, click **Web Module**. On the Web Module configuration page:
 - In the **Context root** field, type the name of your web project root, or accept the default (which is the name of your web project).
 - In the **Content directory** field, type the name of your content directory, or accept the default (`WebContent`).
 - Select **Generate web.xml deployment descriptor** if you want to create a deployment descriptor. You can also add a deployment descriptor to your web module later. You need to use a `web.xml` to configure security constraints and other behavior.
9. Click **Finish**.

Download the address book WSDL file

Download the required WSDL file.

Lesson Checkpoint

Now you are ready to begin [Lesson 2: Creating the web service](#).

Related information:

[FTP download for the sample](#)

[HTTP download for the sample](#)

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 2: Creating the web service

In this lesson you will learn how to create the top-down Java™ address book web service from the WSDL file imported in the previous lesson.

Before you begin, you must complete [Lesson 1: Creating a server and web project](#).

1. Switch to the Java EE perspective: **Window > Open Perspective > Other > Java EE**.
2. In the Enterprise Explorer view, select the WSDL file you imported in the previous lesson.
3. Right-click the WSDL and select **Web Services > Generate Java bean skeleton**.
4. Web Services page: select **Top down Java bean Web service** as your web service type and ensure the `AddressBook.wsdl` is set as the service definition. Choose the following options:
 - A. Select the stages of web services development that you want to complete using the slider. The slider sets the defaults on the remaining wizard pages, but you can override the default settings on each page as you proceed. For this tutorial select **Start service**. This will create all the code required for the web service, deploy it to the server and attempt to start the server.
 - B. Select your server: click the server link and ensure that the WebSphere® v8 Server is selected.
 - C. Select your runtime: ensure the IBM® JAX-WS runtime is selected.
 - D. Select the service project: Select the `jwsAddressBook` project created in the previous lesson.
 - E. Select the service EAR project: Select the `jwsAddressBookEAR` created in the previous lesson.
 - F. The client will be created later, so ensure the client slider is set to **No client**.Click **Next**.
5. WebSphere JAX-WS Top Down Web Service Configuration page:
 - Output folder: Accept the default location where the generated Java skeleton will be generated: `jwsAddressBook/src`.
 - Target package: Accept the default package name
 - Enable wrapper style: Enables wrapper style mapping from WSDL to Java. For WSDL documents that implement a document/literal wrapped pattern, a root element is declared in the XML schema and is used as an operation wrapper for a message flow. Separate wrapper element definitions exist for both the request and the response. Accept the default.
 - Enable MTOM support: If you select this check box the SOAP Message Transmission Optimization Mechanism will be enabled to optimize the transmission of binary content. Accept the default.
 - Version of JAX-WS code to be generated: Starting with WebSphere Application Server v8.0, you can generate JAX-WS 2.1 compliant code. Select 2.1 for this tutorial.
 - Copy WSDL to project: Select this to copy the WSDL file into the service project. Since you will create the client at a later time select this check box.
 - Generate serializable JAXB classes: In WebSphere Application Server v7.0 and up, when you enable the Java 6 facet, you can choose to generate JAXB classes which implement `java.io.Serializable`. Classes that do not implement this interface will not have any of their state serialized or deserialized. Do not select this for the tutorial.
 - Specify JAX-WS or JAXB binding files: this allows you to use JAX-WS or JAXB custom binding files. Do not select this for the tutorial.
 - Customize service implementation class name: this allows you to change the default port name to service implementation class name mapping. Do not select this for the tutorial.
 - Generate schema library: Selecting this will run the JAX-WS Schema to Java compiler to generate a schema. Do not select this option for this tutorial.
 - Generate Web service deployment descriptor: For JAX-WS web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer required. Selecting this check box will generate

them. This tutorial does not require deployment descriptors.

Click **Next**.

All the required code for the web service is generated

Adding the business logic to the skeleton bean

The skeleton implementation bean generated by the web service wizard `AddressBookPortImpl.java` does not contain any business logic. It does contain the annotation `@javax.jws.WebService` which tells the runtime that it is a JAX-WS web service.

In order to make the address book web service function as expected, you need to add code to this bean. It should be opened in an editor automatically after generating the web service, but if not it can be found in: `jwsAddressBook/JavaResources/src/com.addressbook`.

1. Replace the current `saveAddress` method:

```
public boolean saveAddress(PersonType person) {  
    return false;  
}
```

with the following static field and method:

```
private static Hashtable<String,AddressType> addresses = new  
Hashtable<String,AddressType>();  
  
public boolean saveAddress(PersonType person) {  
    addresses.put(person.getName(),person.getAddress());  
    return true;  
}
```

2. Replace the current `findAddress` method:

```
public AddressType findAddress(String name) throws FindAddressFault {  
    return null;  
}
```

with the following:

```
public AddressType findAddress(String name) throws FindAddressFault {  
    return addresses.get(name);  
}
```

3. Several error markers may be displayed. To correct these, organize your imports by clicking `Ctrl+Shift+o`. Select to import `java.util.Hashtable`. After this is done the errors should be grayed out.

4. Save the updated implementation bean.

Test the web service using the Generic Service Client

The Generic Service Client allows you to test a web service without building a client. You can select the operation you want to test, enter the required information, and the result will display in the Status pane.

1. Select the generated WSDL file `jwsAddressBook/WebContent/WEB-INF/wsdl/AddressBook.wsdl`, right-click and select **Web Services > Test with Generic Service Client**. Alternatively you can select the service under the project's Services node or the JAX-WS web services node in the Services view, and launch the Generic Service Client from there.
2. Select the `SaveAddress` operation.
3. Enter values in each of the fields and click **Invoke**.
4. Select the `FindAddress` operation.
5. Enter the name you chose when invoking the `saveAddress` operation and click **Invoke**.
6. The information you saved to this name will display in the Status pane.

Lesson Checkpoint

Now you are ready to begin [Lesson 3: Creating the web service client](#).

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 3: Creating the web service client


In this lesson you will learn how to create a client for the web service.

Before you begin, you must complete [Lesson 2: Creating the web service](#).

Web service clients are created from a WSDL document that describes where the web service is deployed and what operations this service provides. You can use either the static WSDL file generated by the web service wizard, or when creating JAX-WS web services you can use the dynamic WSDL file generated by the runtime based on information it gathers from the annotations added to the Java classes. For this tutorial the dynamic WSDL file will be used.

View the dynamically generated WSDL

If your server is started you should be able to quickly test your annotated bean to ensure it is a web service and generate the dynamic WSDL file.

1. Determine the port your web service is using if you do not already know it. To do this:
 - A. Launch the WebSphere Application Server Administrative Console by right-clicking your server in the Servers view and selecting **Administration > Run administrative console**.
 - B. Expand **Servers > Server Types**, and select **WebSphere application servers**.
 - C. Select your server name from the list. By default this is **server1**.
 - D. On the Configuration tab, search for the Communications heading and expand **Ports**.
 - E. The port used is WC_defaulthost. This tutorial will use the example port number 9081. Replace this number with your own port in any following steps.
2. Launch the Web browser by clicking this icon: 
3. Enter the following URL in the browser: `http://localhost:9081/jwsAddressBook/AddressBookService` and hit return. A page displays stating that this is a web service.
4. Add `?wsdl` to the end of the URL in step 3 and hit return again: `http://localhost:9081/jwsAddressBook/AddressBookService?wsdl`. The dynamically generated WSDL file will be displayed. This file can be used the same way as you would use the static WSDL file that the web service wizard generated as long as the server is running.

Create the address book web service client and test JSP

1. Click **File > New > Other**. Select **Web Services** in order to display the various web service wizards. Select the **Web Service Client** wizard. Click **Next**.
2. Web Services page: Enter the URL of the dynamically generated WSDL file that you discovered in the previous section in the **Service definition** field: `http://localhost:9081/jwsAddressBook/AddressBookService?wsdl`.
 - A. Select the stages of web service client development that you want to complete using the slider. For this tutorial select **Test client**. This will generate all the required client code and provide various testing options for the client.
 - B. Server: Ensure the WebSphere v7.0 and up server is selected.
 - C. Web service runtime: Ensure the JAX-WS runtime is selected.
 - D. Client project: Select the client project created in lesson 1: `jwsAddressBookClient`.
 - E. Client EAR project: Select the `jwsAddressBookEAR`. For JAX-WS web services, the server and client projects can share the same EAR.
 - F. Monitor the web service: This will send the web service traffic through the TCP/IP Monitor, which allows you to watch the SOAP traffic generated by the web service and to test this traffic for WS-I compliance. The TCP/IP Monitor is not capable of monitoring web services secured with the RSP policy set, so do not select this option.

Click **Next**.

3. WebSphere JAX-WS Web Service Client Configuration page:

- Output folder: Accept the default folder where the client's Java classes will be generated.
- Target package: The web services client wizard generates a number of Java files from the specified WSDL. By default it will create a package name based on the namespace specified in the WSDL file. To override this default behavior you can specify your own package name for the namespace in the WSDL file. Accept the defaults for this tutorial.
- Generate portable client: Selecting this checkbox would allow you to move your web service client code from one machine to another or from one instance of WebSphere Application Server to another. It is not required by this tutorial.
- Enable asynchronous invocation for generated client: If you select to enable an asynchronous client, for each method in the web service two additional methods will be created. These are polling and callback methods which allow the client to function asynchronously. Select this option.
- Specify JAX-WS or JAXB binding files: If you have created JAX-WS or JAXB custom binding files, select this check box to use them to create this web service. Do not select this for the tutorial.
- Customize client proxy class name: You can accept the default proxy name or enter your own. Do not select this for the tutorial.
- Generate schema library: Selecting this will run the JAX-WS Schema to Java compiler to generate a schema. Do not select this option for this tutorial.
- Generate web service deployment descriptor: For JAX-WS web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer required. Selecting this checkbox will generate them. This tutorial does not require deployment descriptors.
- Version of JAX-WS code to be generated: Starting with WebSphere Application Server v7.0, you can generate JAX-WS 2.1 compliant code. Select 2.1 for this tutorial.

Click **Next**.

4. Web Service Client Test page:

- Test the generated proxy: Select this to test the functionality of the client.
- Select your test facility. You can test the generated proxy in the Universal Test Client or the Web Service Explorer, or you can generate sample JAX-WS 2.0 JSPs. For this tutorial, select the JAX-WS JSPs.
- Folder: You can select the folder where the JSP will be located. Accept the default.
- Methods: Select the methods to expose. The asynchronous methods should be listed as well. Ensure all methods are selected.
- Run test on server: this will start the server for you automatically.

Click **Finish**. The sample JSP will open in a browser window.

Test the web service client using sample JSPs

The following steps will test the web service synchronously .

1. The `TestClient.jsp` should be launched automatically if you selected the correct options. If it is not displayed, select `jwsAddressBookClient/WebContent/sampleAddressBookPortProxy/TestClient.jsp`, right-click and select **Run As > Run on Server**
2. To test the service synchronously, select the `saveAddress` method and enter information in the name field. All other fields are optional. Click **Invoke**.
3. Select the `findAddress` method, enter the name you used during the `saveAddress` method, and click **Invoke**. The information saved by the `saveAddress` method should display in the results pane.
4. To test the service asynchronously, in the Quality of Service pane select the **Enable asynchronous invocation** checkbox.
5. Select the `findAddress` method, enter the name you used during the `saveAddress` method, and click **Invoke**.
6. A new link will display indicating that the method is in progress. Click the link to display the method response in the Results pane.

Once you have tested the web service leave the JSP open so that you can test the service again once the web service and client have been secured.

Lesson Checkpoint

Now you are ready to begin [Lesson 4: Secure the web service with the RSP policy set](#).

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 4: Attach the WS-I RSP policy set to the web service

In this lesson you will learn how to secure the web service with one of the default policy sets packaged with WebSphere® Application Server.

Before you begin, you must have completed the steps in [Lesson 3: Creating the web service client](#).

You can use the policy sets that are included with this product to simplify configuring the qualities of service for your web services and clients. Several policy sets are included in the workbench. Alternately you can use the administrative console to create your own policy sets and import them.

In this tutorial you will attach the Reliable Secure Profile (RSP) default policy set to the web service and client. The WS-I RSP default policy set consists of instances of the WS-Security, WS-Addressing and WS-ReliableMessaging policy types.

This policy set provides the following features:

- Reliable message delivery to the intended receiver by enabling WS-ReliableMessaging
- Message integrity by digital signature that includes signing the body, timestamp, WS-Addressing headers and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications
- Confidentiality by encryption that includes encrypting the body, signature and signature confirmation elements, using the WS-SecureConversation and WS-Security specifications

1. In the Java™ EE perspective Services view expand the JAX-WS Web Services node. The address book Web service and client should be under their respective folders.
2. Select the address book service, right-click and select **Manage Policy Set Attachment**.
3. Select the jwsAddressBookEAR as the service EAR project and click **Add**.
4. You can apply a policy set at the service, port or operation level. Different policy sets may be applied to various endpoints and operations within a single web service. However the service and client must have the same policy set settings. For this tutorial you will apply the policy set to the entire service, so the Endpoint and Operation Name fields can remain blank.
5. From the Policy Set drop-down list, select **WS-I RSP**, and for the Binding ensure **Provider Sample** is selected. This is a provider-side general binding packaged with WebSphere Application Server. Click **OK**. The service should now be listed in the Application table and the WS-I RSP policy. Click **Finish**.

After a policy set has been attached to a web service a `policyAttachments.xml` file is generated in the EAR META-INF folder. This file will be appended for each additional policy set setting added to any service within the EAR.

Lesson Checkpoint

Now you are ready to begin [Lesson 5: Secure the web service client with the RSP policy set](#).

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Lesson 5: Secure the web service client with the WS-I RSP policy set

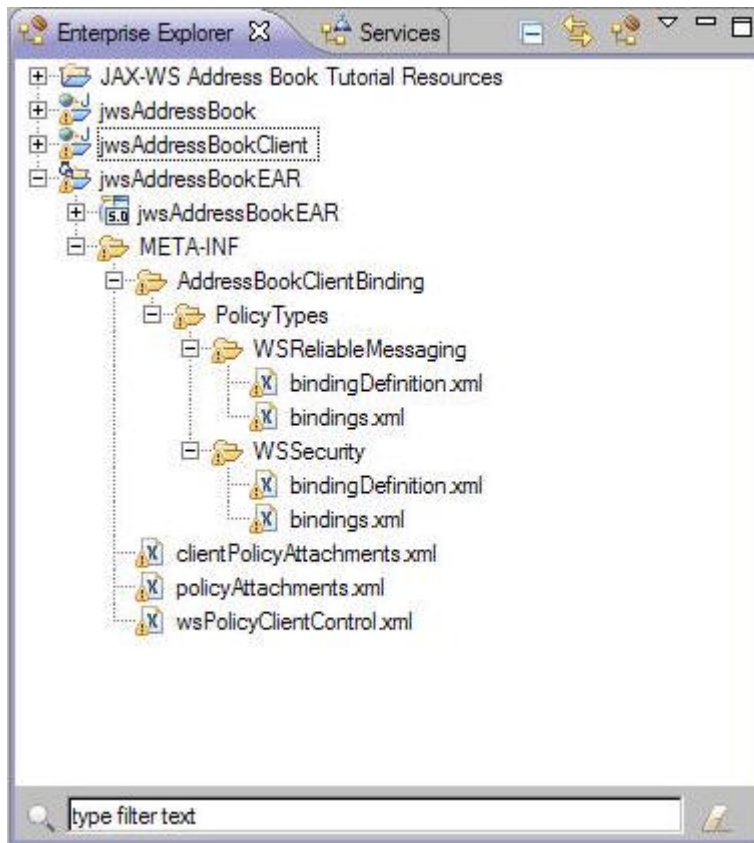
You can add security to a web service client by attaching policy sets to the client. Each attachment specifies an endpoint, a policy set, and a binding. Because each configuration is specific to an application and a user, you must configure a binding for some policy types.

Before you begin, you must have completed the steps in [Lesson 4: Attach the RSP policy set to the web service](#).

For a given web service and a client of that service, the policy sets and bindings configuration must match for the service to function correctly.

1. In the Java™ EE perspective Services view expand the JAX-WS web services node. Select the address book client, right-click, and select **Manage Policy Set Attachment**.
2. Ensure that the `jwsAddressBookEAR` is selected, and click **Next**.
3. In the Application section, click **Add** to attach a policy set to the endpoint and specify the bindings.
 - A. Since the service is secured at the service level rather than the endpoint or operation level, the client will be secured at this level as well. Select the `AddressBookService` from the Service Name drop-down list and leave the Endpoint and Operation Name fields empty.
 - B. In the Policy Set field, select WS-I RSP from the list.
 - C. In the Binding field, ensure **Client Sample** is selected. This is a client-side general binding that is packaged with WebSphere® Application Server.
 - D. Click **OK**.
4. The policy types contained by the policy set you selected are listed in the Bindings Configuration table. The configuration for these policy types are already complete.
5. Click **Finish** to complete the wizard.

`clientPolicyAttachments.xml` is created in the `META-INF` folder of the `jwsAddressBookEAR`, as well as the client side bindings.



Testing the secured web service using the TestClient.jsp

When you ran the `TestClient.jsp` earlier the web service was not secure. The SOAP traffic contained the information being sent to and from the web service in clear (unencrypted) text.

1. Select the `SaveAddress` method, enter information in each of the fields and click **Invoke**.
2. Select the `Find Address` method, enter the name used in the previous step and click **Invoke**. The web service should function in the exact way that it did before it was secured with the information entered in step 1 being displayed in the Results pane.

Lesson Checkpoint

Finish your tutorial by reviewing the materials in the [Summary](#)

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Summary

You have just completed development, securing, and testing of a web service that uses JAX-WS as its runtime environment.

In this tutorial you:

- Created a server and web projects that supported the JAX-WS runtime.
- Used the web service wizard to generate a web service from an imported WSDL document.
- Used the WSDL file dynamically generated by the JAX-WS runtime to create a web service client.
- Tested the web service using the TestClient.jsp test facility using synchronous and asynchronous invocation .
- Attached the RAMP default policy set to the web service.
- Attached and configured the RAMP default policy set to the web service client.
- Tested the secured web service and examined the impact of the policy sets on its function.

When you have completed the tutorial and no longer require the Web projects for testing, remove the EAR from the server, and delete the projects from your workspace.

[< Previous](#) | [Next >](#)

Samples

You can use the available samples to learn about web services and OSGi features in the product.

You can obtain the sample code through HTTP and FTP download from the sample topics. You can get additional samples from the [Liberty Repository](#).

Sample: WebSphere JAX-WS Web service temperature conversion

This sample creates a Java™ EE 5 web service and web service client from an EJB V3.0 Enterprise bean that provides methods to convert Celsius to Fahrenheit and Fahrenheit to Celsius. It uses the WebSphere® JAX-WS runtime environment and runs on WebSphere Application Server V8.5.

The samples are not associated with a server. To run the sample, you must create a Version 8.5 application server and associate the sample with it. For instructions on how to do run the sample, see **Setup instructions**.

When you download the sample, it creates the appropriate service and client projects and EARs that you can test and explore. However, the samples are not associated with a server. To run the sample, you must create a WebSphere Application Server V8.5 and associate the sample with it. For instructions on how to run the sample, review the setup instructions.

To use the sample, download the sample through HTTP or FTP, and save the compressed file to a directory on your workstation. After you download the sample, import the sample project into your workspace, click **File > Import > General > Existing projects into workspace > Select archive file**, and then click **Browse** to find the downloaded sample project file.

- [WebSphere JAX-WS Web service temperature conversion setup instructions](#)

Parent topic: [Samples](#)

Related information:

[Setup instructions](#)

[FTP download for the sample](#)

[HTTP download for the sample](#)

[Tutorial: Creating web services and an EJB skeleton from a WSDL file](#)

WebSphere JAX-WS Web service temperature conversion setup instructions

About this task

To run the sample code, you must create and configure a WebSphere® Application Server V8.5 server and associate the sample with it using the following steps:

Procedure

1. From the **File** menu, select **New > Other > Server > Server > Next**.
2. Select **WebSphere Application Server V8.5** as the server type, and click **Next**.
3. Select a server profile. If no server profile is available, create a profile by clicking **Configure profiles** and running the profile management tool. For more information, see [Creating, editing, and deleting servers](#).
4. Enter `server1` as the server name, and click **Finish**.
5. To see the server you created in the workspace, from the Window menu select **Show view > Other > Server > Servers > OK**.
6. To associate the sample with the server, right-click the server that you created and select **Add and Remove**. Select the service and client EARs from the Available projects list and click **Add**.
7. You should be prompted to restart the server. If not, restart it manually by right-clicking it and clicking **Start**.
8. To run the service, in the Enterprise Explorer right-click the `TemperatureClient/WebContent/sampleConvertTemperatureProxy/TestClient.jsp` and select **Run As > Run on server**.

What to do next

Note: By default the SOAP address in the WSDL file is set to use port 9080. If the port that WebSphere Application Server is using for your service is different, you will need to update the WSDL file to match your port number.

1. You can determine which port WebSphere Application Server is using in the WebSphere administrative console.
 - A. Expand **Servers** and select **Application servers**.
 - B. Select your server name from the list. By default this is **server1**.
 - C. On the Configuration tab under the Communications heading, expand **Ports**. The port that the service uses is `WC_defaulthost`.
2. If the port is anything other than 9080, run the `TestClient.jsp` as described earlier.
3. Run the `getDescription` method of the client. Copy the endpoint value.
4. Under Quality of Service, paste the current endpoint value for the service. Replace the port number with the port that your server uses, and click **Update**.

Parent topic: [Sample: WebSphere JAX-WS Web service temperature conversion](#)

OSGi Counter Service

This sample demonstrates how to declare services in the blueprint file of a bundle, which makes it accessible by other bundles in the application. The OSGi application consists of an OSGi web bundle that contains a servlet that accesses a service that is provided in another bundle project. The application runs on either WebSphere® Application Server full profile or Liberty profile. This sample is an introduction to using OSGi application development tools.

Tip: You can run this sample on WebSphere Application Server Versions 7.0, 8.0, and 8.5. You can also run this sample on WebSphere Application Server Liberty profile.

Learn more about installing WebSphere Application Server Version 7.0: To run this sample on WebSphere Application Server Version 7.0, you must install the Feature Pack for OSGi Applications and Java™ Persistence API 2.0. To install the feature pack:

1. Open the IBM® Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 7.0 Test Environment**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 7.0.
6. In the Features list, ensure that you select **OSGi Applications** under **IBM WebSphere Application Server Version 7.0 Feature Pack for OSGi Applications and Java Persistence API 2.0**.

Learn more about installing WebSphere Application Server Version 8.0: To run this sample on WebSphere Application Server Version 8.0, you must install the server:

1. Open the IBM Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 8.0**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.0.

Learn more about installing WebSphere Application Server Version 8.5: To run this sample on WebSphere Application Server Version 8.5, you must install the server:

1. Open the IBM Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 8.5**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.5.

Learn more about installing WebSphere Application Server Liberty profile: To run this sample on WebSphere Application Server Version Liberty profile, you must install the server. See [Installing Liberty](#) for more information.

To deploy your OSGi Counter application to a server:

1. In Enterprise Explorer, expand **CounterWebBundle** > **CounterWebBundle** > **Servlets**.
2. Right-click **CounterServlet** and select **Run As** > **Run on Server**. The Run On Server dialog opens.
3. Click **WebSphere Application Server** from the list of servers and click **Finish**.

The string `greet.getText()=0 Hello World!` is displayed in the browser. Each time the page is reloaded the value increments.

Switch to the Console view (**Window** > **Show View** > **Console**) to view the output from the server. A successful outcome

displays the output from `CounterImpl.init()`, `GreetImpl.init()`, and `WorldImpl.init()`, based on the initialization method entries for the `CounterImpl`, `GreetImpl`, and `WorldImpl` beans in the blueprint file:[7/21/10 11:42:18:109 EDT]

```
0000002f StepStartBLA A CWWMH0300I: Starting business-level application "WebSphere:blaname=CounterApp".
[7/21/10 11:42:19:421 EDT] 0000002f webapp I com.ibm.ws.webcontainer.webapp.WebGroupImpl WebGroup SRVE0169I:
Loading Web Module: CounterWebBundle.
[7/21/10 11:42:19:515 EDT] 0000002f WASSessionCor I SessionContextRegistry getSessionContext SESN0176I: Will create a new
session context for application key default_host/CounterWebBundle
[7/21/10 11:42:19:531 EDT] 0000002f webcontainer I com.ibm.ws.wswbcontainer.VirtualHost addWebApplication SRVE0250I:
Web Module CounterWebBundle has been bound to default_host[*:9080,*:80,*:9443,*:5060,*:5061,*:443].
[7/21/10 11:42:19:593 EDT] 0000002f StepStartBLA A CWWMH0196I: Business-level application
"WebSphere:blaname=CounterApp" was started successfully.
[7/21/10 11:42:19:640 EDT] 0000001d SystemOut O WorldImpl.init() called.
[7/21/10 11:42:19:640 EDT] 00000032 SystemOut O CounterImpl.init() called.
[7/21/10 11:42:19:687 EDT] 00000032 SystemOut O GreetImpl.init() called.
```

Note: If the output from the `CounterImpl.init()`, `GreetImpl.init()`, and `WorldImpl.init()` does not display in the console output, check the output for error messages during deployment or startup of the application and then check the blueprint files for possible errors in the bean and service definitions.

Parent topic: [Samples](#)

Related information:

[↗ HTTP download for the sample](#)

[↗ FTP download for the sample](#)

[Related Tutorial](#)

OSGi Blog

This sample is an advanced introduction to using OSGi application development tools. The sample demonstrates how to structure the API and implementation code into separate bundles. The OSGi application consists of an OSGi web bundle that contains servlets that access a JPA service provided in another bundle. The application runs on WebSphere® Application Server.

Tip: You can run this sample on WebSphere Application Server Versions 7.0, 8.0, and 8.5. You can also run this sample on WebSphere Application Server Liberty profile.

Learn more about installing WebSphere Application Server Version 7.0: To run this sample on WebSphere Application Server Version 7.0, you must install the Feature Pack for OSGi Applications and Java™ Persistence API 2.0. To Install the feature pack:

1. Open the IBM® Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 7.0 Test Environment**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 7.0.
6. In the Features list, ensure that you select **OSGi Applications** under **IBM WebSphere Application Server Version 7.0 Feature Pack for OSGi Applications and Java Persistence API 2.0**.

Learn more about installing WebSphere Application Server Version 8.0: To run this sample on WebSphere Application Server Version 8.0, you must install the server:

1. Open the IBM Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 8.0**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.0.

Learn more about installing WebSphere Application Server Version 8.5: To run this sample on WebSphere Application Server Version 8.5, you must install the server:

1. Open the IBM Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 8.5**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.5.

Learn more about installing WebSphere Application Server Liberty profile: To run this sample on WebSphere Application Server Version Liberty profile, you must install the server. See [Installing Liberty](#) for more information.

To deploy your OSGi Blog application to a server:

1. Download the sample.
2. Perform the [setup instructions](#).
3. In Enterprise Explorer, expand **com.ibm.ws.eba.example.blog.web** > **BundleContent**.
4. Right-click **index.html** and select **Run As** > **Run on Server**. The Run On Server dialog opens.
5. Click **WebSphere Application Server** from the list of servers and click **Finish**.

The browser opens with the blog application. Before you can post a blog entry, you need to register as an author. Click **Create Author** to register as an author.

- **OSGi Blog setup**

Before you can run this sample on the server, you need to create the data sources and install the blog sample on the server.

- **OSGi Blog details**

Parent topic: [Samples](#)

Related information:

[↗ HTTP community download for the sample](#)

[↗ FTP download for the sample](#)

OSGi Blog setup

Before you can run this sample on the server, you need to create the data sources and install the blog sample on the server.

Procedure

1. If the server is stopped, start the server by running the following steps:
 - A. Switch to the Servers view (**Window > Show View > Servers**).
 - B. Right-click **WebSphere Application Server** and then click **Start**.
2. Open the administrative console. Right-click the server and select **Administration > Run Administrative Console**.
3. Access the JDBC providers view. In the administrative console, in the **Resources** section, expand **JDBC** and click **JDBC providers**.
4. Click the **Derby JDBC Provider** link.
5. Click **Data sources**.
6. Create the data source. Click **New**. In the **JNDI name** field, enter `jdbc/blogdb`. Click **Next**.
7. In the **Database name** field, enter the location of the BLOGDB database. This location must be on the same machine as the WebSphere® Application Server server. If you are deploying on a local server, enter the location of the database in your workspace. To find the location, expand the derby project, right-click the **BLOGDB** folder, and select **Properties**. In the **Resource** section of the properties view, copy the value that is displayed for **Location**. If you are using a remote server, copy the BLOGDB folder from the derby project to a location on the remote server machine. In the **Database name** field, enter the full path of the location of the BLOGDB folder.
8. Go through the rest of the wizard, accept the defaults, and click **Finish** on the last page of the wizard.
9. Click the **Save** link.
10. Test the connection to the data source. In the **Data sources** view, check the box next to the data source that you created and click **Test connection**.

Results

When you have set up the database and data sources, you can [run the sample on server](#).

Parent topic: [OSGi Blog](#)

Removing the blog sample data sources from the server

Procedure

1. Open the WebSphere Application Server Administrative Console by right-clicking your server instance and select **Administration > Run Administrative Console**. The Admin Console opens in the workbench.
2. In the task list, expand **Resources > JDBC** and then select **Data sources**.
3. In the **Data sources** section, check the box next to the data source with the JNDI name `jdbc/blogdb` and then click **Delete**.
4. Save your changes.

OSGi Blog details

The blog application contains the following bundles and projects:

- **com.ibm.json.java**

- This bundle contains the common code for working with JavaScript Object Notation (JSON).

- **com.ibm.ws.eba.example.blog**

- This bundle contains the main application logic code and interacts between the web front end and the back end persistence code layer.

- **com.ibm.ws.eba.example.blog.api**

- This bundle contains the API for the sample.

- **com.ibm.ws.eba.example.blog.app**

- This application project contains the bundles for the sample.

- **com.ibm.ws.eba.example.blog.persistence**

- This bundle contains code that relates to the JPA layer and the interfaces for the main application code to perform functional blog updates, queries, and supplies a comment service.

- **com.ibm.ws.eba.example.blog.web**

- This bundle contains the static web content and backing Java™ code for the web front end of the application.

- **derby**

- This project contains the database and setup script.

Parent topic: [OSGi Blog](#)

OSGi Hello World

This sample demonstrates how to create a simple OSGi application and run it on WebSphere® Application Server. The OSGi application consists of an OSGi web bundle that contains a servlet and an activator that identifies the lifecycle of the bundle in the Console view. This sample is an introduction to using OSGi application development tools.

Tip: You can run this sample on WebSphere Application Server Versions 7.0, 8.0, and 8.5. You can also run this sample on WebSphere Application Server Liberty profile.

Learn more about installing WebSphere Application Server Version 7.0: To run this sample on WebSphere Application Server Version 7.0, you must install the Feature Pack for OSGi Applications and Java™ Persistence API 2.0. To Install the feature pack:

1. Open the IBM® Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 7.0 Test Environment**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 7.0.
6. In the Features list, ensure that you select **OSGi Applications** under **IBM WebSphere Application Server Version 7.0 Feature Pack for OSGi Applications and Java Persistence API 2.0**.

Learn more about installing WebSphere Application Server Version 8.0: To run this sample on WebSphere Application Server Version 8.0, you must install the server:

1. Open the IBM Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 8.0**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.0.

Learn more about installing WebSphere Application Server Version 8.5: To run this sample on WebSphere Application Server Version 8.5, you must install the server:

1. Open the IBM Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 8.5**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.5.

Learn more about installing WebSphere Application Server Liberty profile: To run this sample on WebSphere Application Server Version Liberty profile, you must install the server. See [Installing Liberty](#) for more information.

To deploy your OSGi Hello World application to a server:

1. In Enterprise Explorer, expand **HelloWorld > HelloWorld > Servlets**.
2. Right-click **HelloServlet** and select **Run As > Run on Server**. The Run On Server dialog opens.
3. Click **WebSphere Application Server** from the list of servers and click **Finish**.

The string `Hello OSGi world!` is displayed in the browser.

Parent topic: [Samples](#)

Related information:

[↗ HTTP download for the sample](#)

[↗ FTP download for the sample](#)

OSGi EJB temperature converter sample

This OSGi sample demonstrates an EJB configured as an OSGi bundle and exposed as a service. Detailed information about creating the application that is demonstrated in the sample is contained in the OSGi EJB tutorial.

Important: Applicable edition: Full profile

See the setup instructions for information about how to run the sample.

Related information:

[Setup instructions](#)

[📄 FTP download for the sample](#)

[📄 HTTP download for the sample](#)

[OSGi EJB tutorial](#)

OSGi EJB sample setup information

About this task

This information describes how to set up and run the OSGi EJB temperature conversion sample after you download it.

Checking servers view for the application server

About this task

In this section, you ensure that you have a WebSphere® Application Server v8.5 instance that is set up to run the OSGi application.

Procedure

1. Click the **Servers** tab to access the servers view.
2. Check that you have a WebSphere Application Server v8.5 server. If you do not have a server, right-click in the **Servers** view and select **New > Server**. Choose **WebSphere Application Server v8.5** from the list and follow the rest of the steps of the wizard.
3. Check that the server is running. If the server is running, the status information next to the server says **Started**. If it is not running, to start it, right-click the server and select **Start**.

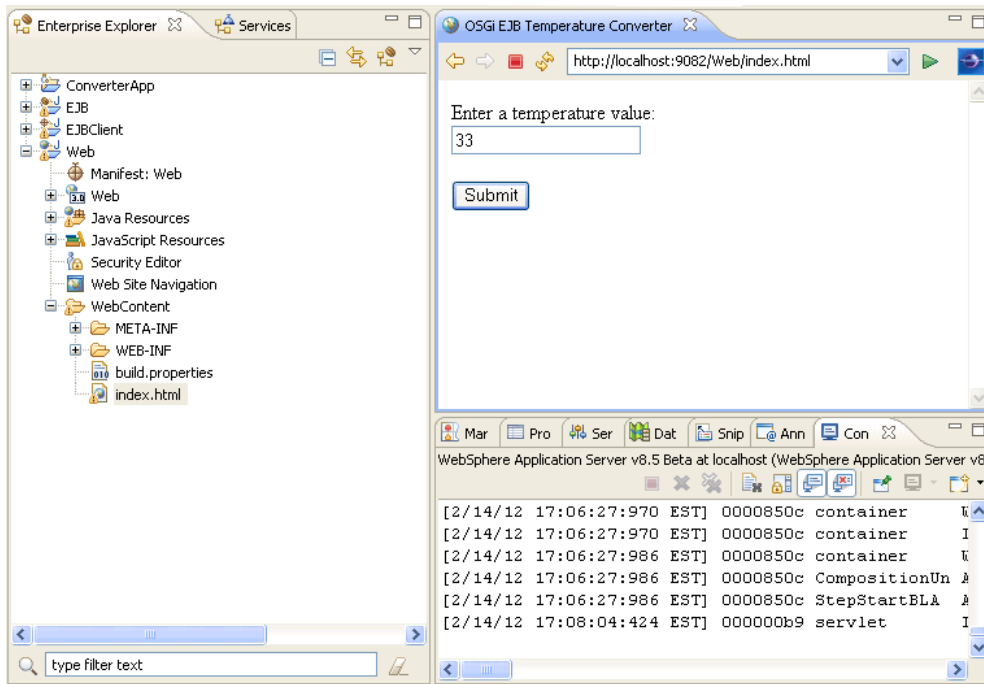
Running the application on the server

About this task

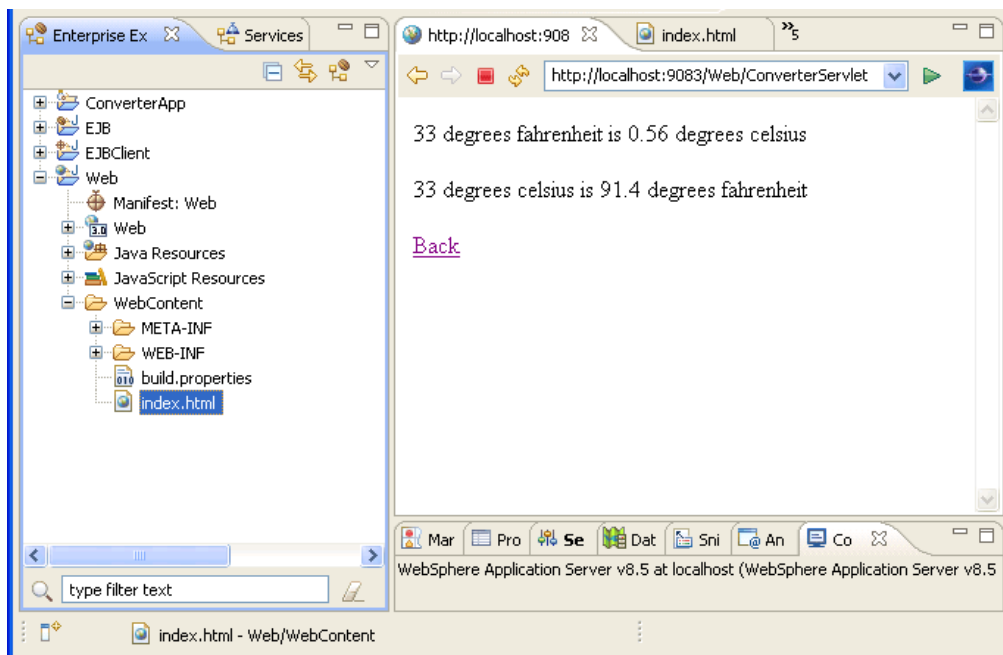
In this section, you run the application and submit test values.

Procedure

1. Launch the application through the `index.html` file. In the web project, right-click the `index.html` file that you created and select **Run As > Run on Server**. Choose the WebSphere Application Server v8.5 server from the server list and click **Finish**. The application is deployed to the server and the deployed `index.html` file opens in the embedded browser.
2. In the field on the page, enter a temperature value that you want to convert.



3. Click **Submit**. The value is converted. The processing is handled by the EJBCConverter EJB that is available as an OSGi service.



OSGi Calculator

The sample demonstrates how to use Maven in OSGi bundles. The OSGi application consists of an OSGi web bundle that contains a servlet that access services provided by other bundle projects. The bundle projects that provide the services (addition, multiplication) are Maven projects that are converted to OSGi. The application runs on WebSphere® Application Server. This sample is an introduction to using OSGi development tools.

To use this sample, you must have WebSphere Application Server version 8.0 or later installed and configured. To verify that a server runtime environment is available, click **Window > Preferences**, expand **Server**, and then click **Runtime Environments**. You can use this pane to add, remove, or edit installed server runtime definitions. You can also download and install support for a new server.

To deploy the sample:

1. Import the sample.
2. In the Enterprise Explorer view, expand **org.sample.calculator.web > org.sample.calculator.web > Servlets**.
3. Right-click **org.sample.calculator.web.CalculatorServlet** and select **Run on Server**.
4. Select WebSphere Application Server version 8.0 or later.

Parent topic: [Samples](#)

Related information:

[↗ HTTP download for the sample](#)

[↗ FTP download for the sample](#)

Installing WebSphere Application Server Developer Tools for Eclipse

IBM® WebSphere® Application Server Developer Tools for Eclipse is a lightweight set of tools for developing, assembling, and deploying Java™ EE, OSGi, Web 2.0, and mobile applications to WebSphere Application Server.

About this task

To install WebSphere Application Server Developer Tools for Eclipse, refer to the following topics:

- **[System requirements for WebSphere Application Server Developer Tools for Eclipse](#)**

Before you install the software, ensure that your computer meets the requirements.

- **[Installing and updating into an existing Eclipse workbench](#)**

You can install IBM WebSphere Application Server Developer Tools for Eclipse into an existing Eclipse workbench from Eclipse Marketplace (**Help > Eclipse Marketplace**) or from downloaded installation files. You can also update IBM WebSphere Application Server Developer Tools for Eclipse installed into an existing Eclipse workbench.

- **[Creating a runtime environment with an optional IBM SDK](#)**

When you install WebSphere Application Server beginning with Version 8.5, by default a specific version of IBM SDK, Java Technology Edition is installed with it. After you install WebSphere Application Server, you can optionally create a runtime environment with other versions of the IBM SDK that the application server supports.

- **[Converting an unlicensed version of WebSphere Application Server Developer Tools for Eclipse into a licensed version](#)**

If you installed the product from Eclipse Marketplace but you later purchased a license for the product (for example, you purchased a WebSphere Application Server bundle that includes IBM WebSphere Application Server Developer Tools for Eclipse), then you must install a supported license.

- **[Known problems and limitations for WebSphere Application Server Developer Tools for Eclipse, Version 8.5.5](#)**

Review the known problems and limitations in WebSphere Application Server Developer Tools for Eclipse, Version 8.5.5.

What to do next

To learn more about WebSphere Application Server Liberty Profile, see [The Liberty profile](#) topic available in the Information Center for WebSphere Application Server.

System requirements for WebSphere Application Server Developer Tools for Eclipse

Before you install the software, ensure that your computer meets the requirements.

Detailed information on system requirements and supported platforms

For detailed information on system requirements and supported platforms for IBM® WebSphere® Application Server Developer Tools for Eclipse, go to <http://pic.dhe.ibm.com/infocenter/prodguid/v1r0/clarity/softwareReqsForProduct.html> and search on the product name.

Requirements for installing into an existing Eclipse workbench

If you choose to install IBM WebSphere Application Server Developer Tools for Eclipse into an Eclipse workbench that is already installed on your computer, then your Eclipse workbench must be Eclipse IDE for Java™ EE Developers version:

- **8.5.5.5** 4.4.2
- **8.5.5.4** 4.4.1
- **8.5.5.2** 4.4.0
- **8.5.5.2** 4.3.2
- **8.5.5.2** 4.3.1
- or equivalent

Runtime environment for Java (JRE) requirements

The required JRE depends on the features that you are installing: *Table 1. Required JRE. This table lists the required JRE for product features.*

If you are installing any of the WebSphere Application Server Tools features for versions 7.0 and later.	If you are NOT installing any of the WebSphere Application Server Tools features for versions 7.0 and later.
---	---

<p>IBM Runtime Environment for Windows, Java Technology Edition, Version 6.0 SR9 FP1 or later. IBM Runtime Environment for Linux, Java Technology Edition, Version 6.0 SR9 FP1 or later. IBM Runtime Environment for Windows, Java Technology Edition, Version 7.0 GA or later. IBM Runtime Environment for Linux, Java Technology Edition, Version 7.0 GA or later. Tip: The IBM Runtime Environment for Windows or Linux, Java Technology Edition Version 6.0 is included with WebSphere Application Server version 8.5, 8.0 and version 7.0. It is located in the following directory <i><WebSphere Application Server installation directory>/java/jre</i></p> <p>Tip: The IBM Runtime Environment for Windows or Linux, Java Technology Edition Version 7.0 is optionally installed with WebSphere Application Server version 8.5. If installed, it is located in the following directory: <i><WebSphere Application Server installation directory>/java_1.7_32/jre</i></p>	<p>8.5.5.5 Any Java 7+ or Java 8+ runtime environment The following versions work well:</p> <p>8.5.5.6 IBM Runtime Environment for Windows or Linux, Java Technology Edition, Version 8.0 SR 1 or later. IBM Runtime Environment for Windows or Linux, Java Technology Edition, Version 7.0 SR 9 or later. IBM Runtime Environment for Windows or Linux, Java Technology Edition, Version 6.0 SR16 FP4 or later. Oracle Java Platform Standard Edition Runtime Environment Version 8.0 latest Update available. Tip: IBM Developer Kits and Runtime Environments are available at: http://www.ibm.com/developerworks/java/jdk/</p>
--	---







Parent topic: [Installing WebSphere Application Server Developer Tools for Eclipse](#)

Installing and updating into an existing Eclipse workbench

You can install IBM® WebSphere® Application Server Developer Tools for Eclipse into an existing Eclipse workbench from Eclipse Marketplace (**Help > Eclipse Marketplace**) or from downloaded installation files. You can also update IBM WebSphere Application Server Developer Tools for Eclipse installed into an existing Eclipse workbench.

Before you begin

- Ensure that your computer meets the [system requirements](#).
- Your Eclipse workbench must be Eclipse IDE for Java™ EE Developers version:

-  4.5
-  4.4.2
-  4.4.1
-  4.4.0
-  4.3.2
-  4.3.1

- or equivalent

For more information, see the [system requirements](#).

- If you have an earlier version of IBM WebSphere Application Server Developer Tools for Eclipse older than 8.5.5 installed (e.g. 8.0.4), you cannot update to version 8.5.5 or later. Uninstall all features of the earlier version and then install version 8.5.5 or later into your Eclipse workbench.

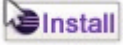
- The developer tools for WebSphere Application Server version 7.0 and later are supported on Windows and Linux operating systems only. If you attempt to install these tools on another type of operating system, then you will get an error similar to the following example during the installation:Cannot complete the install because some dependencies are not satisfiable

```
com.ibm.websphere.wdt.st.v7.feature.group ____ cannot be installed in this environment because its filter is not applicable
```

If you install the developer tools for WebSphere Application Server version 7.0 or later on a form of Linux operating system that is not supported (for example, Ubuntu), you will not get an error message during the installation, but the tools will not work.

About this task

You can install the software using any of the following methods:

- [Install from a remote repository](#).
 - Find and install the remote installation files from your Eclipse workbench. Use your workbench to locate and install the installation files for WebSphere Application Server Developer Tools for Eclipse V8.5.5 and later from Eclipse Marketplace.
 - Drag an Install icon from the Eclipse Marketplace or WASdev Community web page to your workbench. From the Eclipse Marketplace web page (<http://marketplace.eclipse.org/>) or the WASdev community download page (<https://www.ibm.com/developerworks/mydeveloperworks/blogs/wasdev/entry/download>), you can start the installation of WebSphere Application Server Developer Tools for Eclipse V8.5.5 and later by dragging the Install icon () for the tools version that you want to install from your web browser to your running workbench.
- [Install from downloaded installation files](#). Use this method if you want to download the installation files for WebSphere Application Server Developer Tools for Eclipse V8.5.5 and later and then install the product into an existing Eclipse workbench.

Installing from a remote repository

Before you begin

Ensure that your computer is connected to the Internet.


About this task

You can install the product from installation files in a remote repository by either using your Eclipse workbench find the installation files or by finding a link to the installation files on a web page.

The developer tools are available for the following versions of WebSphere Application Server:

- IBM WebSphere Application Server Liberty Profile Developer Tools
- IBM WebSphere Application Server V8.5 Developer Tools
- IBM WebSphere Application Server V8.0 Developer Tools
- IBM WebSphere Application Server V7.0 Developer Tools

Procedure

1. Start your Eclipse workbench.
2. Start the installation using either of the following methods.
 - **Locate the installation files from your Eclipse workbench:**
 - A. Click **Help > Eclipse Marketplace**.
 - B. In the Find field, type `webSphere`.
 - C. In the list of results, locate **IBM WebSphere Application Server *version* Developer Tools V8.5.5** (and later) and then click **Install**. The Confirm Selected Features page opens.
 - **Drag an Install icon from a web page to your Eclipse workbench:**
 - A. Open your web browser to <http://www.ibm.com/wasdev> and click the **Download** tab.
 - B. Locate the **Install** icon () for IBM WebSphere Application Server *version* Developer Tools.
 - C. Select and drag the **Install** icon to your Eclipse workbench, and drop it on the Eclipse toolbar. The Confirm Selected Features page opens.
3. On the **Confirm Selected Features** page, expand the parent nodes and select the features that you want to install. When you are finished, click **Next**.
4. On the Review Licenses page, review the license text. If you agree to the terms, click **I accept the terms of the license agreement** and then click **Finish**. The installation process starts. **Note:** During the installation, a Security Warning dialog box might open and display the following message: Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the installation?
You can safely ignore the message and click **OK** to continue.
5. When the installation process completes, restart the workbench.

What to do next

[Configure the installation](#)

Installing from downloaded installation files

You can download installation files and, if needed, install the downloaded installation files while not connected to the internet.

Before you begin

If you are not connected to the internet during the installation, then you must first download and install prerequisite Eclipse files before you install WebSphere Application Server Developer Tools for Eclipse.

- **8.5.5.5** For Eclipse Luna, complete the following steps:

1. Download the following prerequisite files for Version 4.4.2:
 - [emf-transaction-runtime-1.8.0.zip](#)
 - [emf-validation-runtime-1.8.0.zip](#)
 - [gmf-runtime-1.8.0.zip](#)
2. For each compressed file that you downloaded in the previous step, extract the contents of the file into the Eclipse directory of your Eclipse IDE for Java EE Developers. Ensure that you preserve the structure of the extracted contents. For example, if your Eclipse directory is located in the file path `C:\eclipse-jee-luna-SR1-win32-x86_64\`, then you must extract the contents of each file into the directory `C:\eclipse-jee-luna-SR1-win32-x86_64\`.

Note: You can overwrite any files that have the same name.

When you are finished, the extracted files will be in the existing directories of your Eclipse IDE:

- eclipse (For example, `C:\eclipse-jee-luna-SR1-win32-x86_64\eclipse`.)
- eclipse\plugins (For example, `C:\eclipse-jee-luna-SR1-win32-x86_64\eclipse\plugins`.)
- eclipse\features (For example, `C:\eclipse-jee-luna-SR1-win32-x86_64\eclipse\features`.)

3. From a command line, restart your Eclipse IDE for Java EE Developers using the **-clean** option. For example,
C:\eclipse-jee-luna-SR1-win32-x86_64\eclipse\eclipse.exe -clean

- **8.5.5.6** For Eclipse Mars, complete the following steps:

1. Download the following prerequisite files for Version 4.5:
 - [emf-transaction-runtime-1.9.0.zip](#)
 - [emf-validation-runtime-1.9.0.zip](#)
 - [gmf-runtime-1.9.0.zip](#)
2. For each compressed file that you downloaded in the previous step, extract the contents of the file into the Eclipse directory of your Eclipse IDE for Java EE Developers. Ensure that you preserve the structure of the extracted contents. For example, if your Eclipse directory is located in the file path `C:\eclipse-jee-mars-SR1-win32-x86_64\`, then you must extract the contents of each file into the directory `C:\eclipse-jee-mars-SR1-win32-x86_64\`.

Note: You can overwrite any files that have the same name.

When you are finished, the extracted files will be in the existing directories of your Eclipse IDE:

- eclipse (For example, `C:\eclipse-jee-mars-SR1-win32-x86_64\eclipse`.)
- eclipse\plugins (For example, `C:\eclipse-jee-mars-SR1-win32-x86_64\eclipse\plugins`.)
- eclipse\features (For example, `C:\eclipse-jee-mars-SR1-win32-x86_64\eclipse\features`.)

3. From a command line, restart your Eclipse IDE for Java EE Developers using the **-clean** option. For example,
C:\eclipse-jee-mars-SR1-win32-x86_64\eclipse\eclipse.exe -clean

About this task

Perform the following steps to install the product into an existing Eclipse workbench from downloaded installation files.

Procedure

1. Download the .zip file of IBM WebSphere Application Server Developer Tools for Eclipse to a directory on your computer. The .zip file is available at:

<https://www.ibm.com/developerworks/mydeveloperworks/blogs/wasdev/entry/download>.

2. Start your Eclipse workbench.
3. Click **Help > Install new software > Add**.
4. In the Add Repository window, click **Archive**.
5. Browse to the location of the .zip file of IBM WebSphere Application Server Developer Tools for Eclipse. Select the file and then click **Open**.
6. In the list of results, select **IBM WebSphere Application Server Developer Tools for Eclipse V8.5.5** (and later) . If you do not want to install all features (for example, you are installing only the WebSphere Application Server V8.5 Liberty Profile Tools feature), then expand the **IBM WebSphere Application Server Developer Tools for Eclipse V8.5.5** (and later) node and select the features that you want to install.**Tip:** Ensure that you do not select **IBM WebSphere Application Server Liberty Profile Server Adapter**.

You can install the following features:

Feature	Description
Dojo Toolkit (optional)	Provides Dojo Toolkit for building Web 2.0 applications. Important: If you are installing from Eclipse Marketplace, then this option is not listed. If you are installing from the Eclipse Marketplace and want install Dojo Toolkit:Install IBM WebSphere Application Server Developer Tools for Eclipse.Start the product.Click Help > Install new software .In the Work with field, select All Available Sites from the list.Expand the WebSphere Application Server Developer Tools for Eclipse category and select Dojo Toolkit .Follow the prompts in the wizard to install the software.
OSGi Application Development Tools	Provides tools for developing and maintaining OSGi-based applications and bundles.
Web Development Tools	Develop web applications using the latest web technologies such as HTML5, CSS3, Dojo Toolkit, Servlet 3.0 and JSP 2.2.
Web Services Development Tools - Liberty	Tools for developing and testing JAX-WS Web services and clients for Liberty.
WebSphere Application Server - Liberty Profile	Tools for developing and administering WebSphere Application Server - Liberty Profile.
WebSphere Application Server V8.5 Tools	Tools for developing and administering WebSphere Application Server V8.5.
WebSphere Application Server V8.0 Tools	Tools for developing and administering WebSphere Application Server V8.0.
WebSphere Application Server V7.0 Tools	Tools for developing and administering WebSphere Application Server V7.0.

7. Click **Next**.

8. On the Review Licenses page, review the license text. If you agree to the terms, click **I accept the terms of the license agreement** and then click **Finish**. The installation process starts. **Note:** During the installation, a Security Warning dialog box might open and display the following message: Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the installation?

You can safely ignore the message and click **OK** to continue.

9. When the installation process completes, restart the workbench.

What to do next

1. [Configure the installation.](#)

2. If you purchased a license for the product (for example, you purchased a WebSphere Application Server bundle that includes WebSphere Application Server Developer Tools for Eclipse), then you must install a supported license.

Configuring the installation About this task

After you install the software, you must complete the following required configuration steps. **Attention:** These configuration steps require you to modify files. Create a backup copy of the files before you modify them.

- Required configuration steps:

1. Stop the workbench.
2. Locate the `eclipse.ini` file in the `<Eclipse installation directory>\eclipse` directory, where `<Eclipse installation directory>` is the directory where you installed Eclipse.
3. Make a copy of the `eclipse.ini` file, or back it up.
4. Open the `eclipse.ini` file in a text editor.
5. Locate the line `-vmargs` and add the following text immediately after it according to the following cases:

- **If you are using an IBM 32-bit JRE**, then add the following text after the `-vmargs` line: `-Xms100m`

```
-Xmx1024m
-Xscmx48m
-Xshareclasses:name=IBMSDP_%u
-Xquickstart
-Xgcpolicy:gencon
-Xmnx64m
-Dcom.ibm.ws.management.event.max_polling_interval=1000
```

- **If you are using an IBM 64-bit JRE**, then add the following text after the `-vmargs` line: `-Xms100m`

```
-Xmx1024m
-Xscmx48m
-Xshareclasses:name=IBMSDP_%u
-Xcompressedrefs
-Xquickstart
-Xgcpolicy:gencon
-Xmnx64m
-Dcom.ibm.ws.management.event.max_polling_interval=1000
```

- **If you are using an Oracle 32-bit JRE**, then add the following text after the `-vmargs` line: `-Xms100m`

```
-Xmx960m
-XX:MaxPermSize=320m
```

```
-Dcom.ibm.ws.management.event.max_polling_interval=1000
```

- **If you are using an Oracle 64-bit JRE**, then add the following text after the `-vmargs` line: `-Xms100m`

```
-Xmx1024m
```

```
-XX:MaxPermSize=512m
```

```
-Xmx1024m
```

```
-XX:+UseCompressedOops
```

```
-Dcom.ibm.ws.management.event.max_polling_interval=1000
```

6. Locate the line `--launcher.XXMaxPermSize` and in the `eclipse.ini` file and modify the value:

- **If you are using an Oracle 32-bit JRE**, then change the value to `320M` as shown in the following text:--

```
launcher.XXMaxPermSize
```

```
320M
```

- **If you are using an Oracle 64-bit JRE**, then change the value to `512M` as shown in the following text:--

```
launcher.XXMaxPermSize
```

```
512M
```

7. **If you installed any of the WebSphere Application Server Tools features**, insert the following text immediately

before the line with the text `-vmargs`:

```
- Windows : -vm
```

```
<WebSphere Application Server V7.0 and later installation directory>\java\jre\bin\javaw.exe
```

```
- Linux : -vm
```

```
<WebSphere Application Server V7.0 and later installation directory>/java/jre/bin/java
```

8. Save and close the `eclipse.ini` file.

9. Restart the workbench.

8.5.5.2 Procedure

1. Stop the workbench.

2. Locate the `eclipse.ini` file in the `<Eclipse installation directory>\eclipse` directory, where `<Eclipse installation directory>` is the directory where you installed Eclipse.

3. Make a copy of the `eclipse.ini` file, or back it up.

4. Open the `eclipse.ini` file in a text editor.

5. Locate the line `-vmargs` and add the following text immediately after it according to the following cases:

- **If you are using an IBM 32-bit JRE**, then add the following text after the `-vmargs` line: `-Xms100m`

```
-Xmx1024m
```

```
-Xscmx48m
```

```
-Xshareclasses:name=IBMSDP_%u
```

```
-Xquickstart
```

```
-Xgcpolicy:gencon
```

```
-Xmnx64m
```

```
-Dcom.ibm.ws.management.event.max_polling_interval=1000
```

- **If you are using an IBM 64-bit JRE**, then add the following text after the `-vmargs` line: `-Xms100m`

```
-Xmx1024m
```

```
-Xscmx48m
```

```
-Xshareclasses:name=IBMSDP_%u
```

```
-Xcompressedrefs
```

```
-Xquickstart
```

```
-Xgcpolicy:gencon
```

```
-Xmnx64m
```

```
-Dcom.ibm.ws.management.event.max_polling_interval=1000
```

- **If you are using an Oracle 32-bit JRE**, then add the following text after the `-vmargs` line:`-Xms100m`

```
-Xmx960m  
-XX:MaxPermSize=320m  
-Dcom.ibm.ws.management.event.max_polling_interval=1000
```

- **If you are using an Oracle 64-bit JRE**, then add the following text after the `-vmargs` line:`-Xms100m`

```
-Xmx1024m  
-XX:MaxPermSize=512m  
-Xmx1024m  
-XX:+UseCompressedOops  
-Dcom.ibm.ws.management.event.max_polling_interval=1000
```

6. Locate the line `--launcher.XXMaxPermSize` in the `eclipse.ini` file and modify the value:

- **If you are using an Oracle 32-bit JRE**, then change the value to `320M` as shown in the following text:--

```
launcher.XXMaxPermSize  
320M
```

- **If you are using an Oracle 64-bit JRE**, then change the value to `512M` as shown in the following text:--

```
launcher.XXMaxPermSize  
512M
```

7. **If you installed any of the WebSphere Application Server Tools features**, insert the following text immediately before the line with the text `-vmargs`:

```
- Windows: -vm  
<WebSphere Application Server V7.0 and later installation directory>\java\jre\bin\javaw.exe  
- Linux: -vm  
<WebSphere Application Server V7.0 and later installation directory>/java/jre/bin/java
```

8. Save and close the `eclipse.ini` file.

9. Restart the workbench.

8.5.5.2 Creating a runtime environment with an optional IBM SDK

When you install WebSphere® Application Server beginning with Version 8.5, by default a specific version of IBM® SDK, Java™ Technology Edition is installed with it. After you install WebSphere Application Server, you can optionally create a runtime environment with other versions of the IBM SDK that the application server supports.

Before you begin

Important: Applicable edition: Full profile

- Ensure that IBM WebSphere Application Server Developer Tools for Eclipse is already installed into your Eclipse IDE for Java EE Developers workbench and the workbench has been restarted. For installation instructions, see [Installing and updating into an existing Eclipse workbench](#).
- If you are not installing from the Eclipse marketplace, and do not already have the installation files (.zip file) for IBM WebSphere Application Server Developer Tools for Eclipse, download the file from <https://www.ibm.com/developerworks/mydeveloperworks/blogs/wasdev/entry/download>.

About this task

WebSphere Application Server Version 8.5 provides support for IBM SDK Version 7.0 and IBM SDK Version 7.1. These IBM SDKs are optional. After you install WebSphere Application Server Version 8.5, you can install either IBM SDK on it. Earlier versions of WebSphere Application Server, for example Version 7.0 and Version 8.0, do not support optional IBM SDKs. When you move from either of these earlier versions to WebSphere Application Server Version 8.5, you can continue running your existing applications on Version 6 of the IBM SDK. However, you can instead run them on Version 7.0 or Version 7.1 of the IBM SDK.

The following table summarizes what versions of the IBM SDK apply to a particular version of WebSphere Application Server. The table also summarizes what versions of the IBM SDK can optionally be used to create an application server runtime environment *Table 1. WebSphere Application Server versions and their IBM SDK versions*

WebSphere Application Server versions	Supported IBM SDK versions	IBM SDKs that can optionally be used to create an application server runtime environment
7.0	6.0	None
8.0	6.0	None
8.5	6.0, 7.0, 7.1	7.0 and 7.1

The following example procedure shows you how to create a WebSphere Application Server Version 8.5 runtime environment with IBM SDK, Java Technology Edition, Version 7.1.

Procedure

1. In the workbench, select **Window > Preferences > Server > Runtime Environments**.
2. Click **Add**.
3. Select **WebSphere Application Server v8.5**, and then click **Next**.
4. Specify the installation directory. The **JRE for the runtime environment:** list is automatically displayed. The list contains the IBM SDKs including the default IBM SDK, which is Version 6.

5. Select Version 7.1, and then click **Finish**.

Results

You have an application server that is targeted for JRE 1.7.1.

Parent topic: [Installing WebSphere Application Server Developer Tools for Eclipse](#)

Converting an unlicensed version of WebSphere Application Server Developer Tools for Eclipse into a licensed version

If you installed the product from Eclipse Marketplace but you later purchased a license for the product (for example, you purchased a WebSphere® Application Server bundle that includes IBM® WebSphere Application Server Developer Tools for Eclipse), then you must install a supported license.

Before you begin

Download the compressed file of the license of WebSphere Application Server Developer Tools for Eclipse from IBM Passport Advantage®. The license file is named `wdt-supported-license_<version>.zip`

Procedure

1. Start the WebSphere Application Server Developer Tools for Eclipse workbench.
2. Click **Help > Install New Software**.
3. On the Available Software page, click **Add**.
4. In the Add Repository window, click **Archive**.
5. Browse to the location where you downloaded the compressed file of the license of WebSphere Application Server Developer Tools for Eclipse. Select the file **wdt-supported-license_<version>.zip** and then click **Open**.
6. In the **Name** field, select **IBM WebSphere Application Server Developer Tools for Eclipse** and then click **Next**.
7. Click **Next** on the Install Details Page.
8. On the Review Licenses page, review the license text. If you agree to the terms, click **I accept the terms of the license agreement** and then click **Finish**. The installation process starts.
9. When the installation process completes, restart the workbench.

Parent topic: [Installing WebSphere Application Server Developer Tools for Eclipse](#)

Known problems and limitations for WebSphere Application Server Developer Tools for Eclipse, Version 8.5.5

Review the known problems and limitations in WebSphere® Application Server Developer Tools for Eclipse, Version 8.5.5.

Contents

- [Known problems with workarounds](#)

- [Known limitations](#)

Known problems with workarounds

Component	Problem description and workaround
JPA Tools	<p>NPE when creating entities from JPA Manager Bean or Configure Entities Wizard for non-JPA projects When a user attempts to create Entities from the JPA Manager Bean Wizard or Configure Entities Wizard for a project that doesn't have the JPA facet installed, a Null Pointer Exception is thrown.</p> <p>To work around the problem: Open project Properties. Go to Project Facets. Check JPA facet. Click OK.</p>
Liberty tools	<p>To start from the workbench an enterprise application with EJB modules published to WebSphere Application Server Liberty Profile, you must enable the Run applications directly from workspace publish option in the server editor. Otherwise, the application cannot be started.</p>
Maven tools	<p>Target runtime server dependencies cannot be added to the POM, and a warning occurs. A validation verifies the configuration of the project checks and the content of the POM file associated with the target runtime server dependencies. However, currently it is not possible to declare target runtime server dependencies in the POM.</p> <p>To work around the problem: You can stop the warning for the project by removing the target runtime from the project properties.</p>
Mobile Browser Simulator	<p>The Mobile Browser simulator opens with iPhone 4 as the default. Changing the device to an Android device has no effect.</p> <p>To work around the problem: You can preview only in iPhone mode, or test on an actual device.</p>
OSGi Tools	<p>In an OSGi bundle that is configured with Maven support, after removing a header element from the instructions section in the <code>pom.xml</code> file of the maven-bundle-plugin, the header is still in the manifest after the file is regenerated.</p> <p>To work around the problem: Manually remove the unwanted header from the manifest.</p>

OSGi Tools	<p>When creating a blueprint file in an OSGi bundle that is configured with Maven support, the wizard does not add a Bundle-Blueprint header to the <code>pom.xml</code> file. This omission is a problem only if the blueprint file is created in a non-standard location (a location other than <code>OSGI-INF/blueprint/</code>).</p> <p>To work around the problem: Accept the default location when creating blueprint files in OSGi bundles that are configured with Maven support.</p>
Rich Page Editor	<p>Rich Page Editor is supported on Ubuntu Linux, with the following considerations:Installation requirements for embedded browsers on Linux systemsScrollbars in Rich Page Editor do not function on Ubuntu 12.04</p>
Web services	<p>Using the test client JSP, if you select Enable asynchronous invocation and then invoke the service, the following error message displays: The Response Class is not defined.</p> <p>To work around the problem: Test the service synchronously.</p>
Web services	<p>Liberty does not support publishing to UDDI. However, Publish the Web service is still selected in the Web service wizard.</p> <p>To work around the problem: Clear the Publish the Web service check selection. The Web Service Publication page also still appears; clear all check boxes on this page.</p>
Web services	<p>Liberty does not support Thin Clients. However, the Web service wizard does not filter out Java™ projects as valid choices to generate JAX-WS Web service clients on Liberty.</p> <p>To work around the problem: Use a different project type, such as a Web project.</p>
Web services	<p>If you target the Liberty profile and you are generating a bottom up JAX-WS Web service using the Web service wizard, a Null Pointer Exception occurs.</p>

Known limitations

Component	Limitation
-----------	------------

General	<p>The product might fail to start on some older Linux platforms. On some older Linux platforms, such as RHEL 4, or SLES 10.4, the product might not start correctly. Shortly after choosing a workspace location, the product will fail with a generic error message. If you inspect the java core file, you see a General Protection Fault in the module: /lib/libc.so.6.</p> <p>For example: NULL</p> <pre> ----- 0SECTION TITLE subcomponent dump routine NULL ===== 1TISIGINFO Dump Event "gpf" (00002000) received 1TIDATETIME Date: 2012/11/16 at 15:14:57 1TIFILENAME Javacore filename: /root/javacore.20121116.151456.12391.0002.txt 1TIREQFLAGS Request Flags: 0x81 (exclusive+preempt) 1TIPREPSTATE Prep State: 0x0 1TIPREPINFO Exclusive VM access not taken: data may not be consistent across javacore sections NULL ----- 0SECTION GPINFO subcomponent dump routine NULL ===== 2XHOSLEVEL OS Level : Linux 2.6.16.60-0.85.1- bigsm 2XHCPU Processors - 3XHCPUARCH Architecture : x86 3XHNUMCPUS How Many : 2 3XHNUMASUP NUMA is either not supported or has been disabled by user NULL 1XHEXCPCODE J9Generic_Signal_Number: 00000004 1XHEXCPCODE Signal_Number: 0000000B 1XHEXCPCODE Error_Value: 00000000 1XHEXCPCODE Signal_Code: 00000001 1XHEXCPCODE Handler1: B758949B 1XHEXCPCODE Handler2: B755E915 1XHEXCPCODE InaccessibleAddress: 00000000 NULL 1XHEXCPCMODULE Module: /lib/libc.so.6 1XHEXCPCMODULE Module_base_address: B7634000 1XHEXCPCMODULE Symbol: memmove 1XHEXCPCMODULE Symbol_address: B769F470 </pre> <p>Work around the problem, as follows: Edit (or create) the <i>workspace/.metadata/.plugins/org.eclipse.core.runtime/.settings/org.eclipse.e4.ui.css.swt.theme.prefs</i> file. Add the preference to enable the classic theme by adding the following lines: eclipse.preferences.version=1 themeid=org.eclipse.e4.ui.css.theme.e4_classic</p>
Web services	The Generic Service Client is not supported on OS X. Use the WSE or sample JSPs to test your Web service.
Web services	Liberty does not support publishing to UDDI.
Web services	Liberty does not support Thin Clients.

Parent topic: [Installing WebSphere Application Server Developer Tools for Eclipse](#)

Developing enterprise applications

You can develop enterprise applications by using WebSphere Developer Tools. You can create Java EE applications, enterprise beans, Java Persistence API applications, and WebSocket applications.

- [Learn about enterprise applications](#)

The workbench provides the tools you need to develop enterprise applications. You can use the Java EE tools and features to create applications that are structured around modules with different purposes, such as web sites and enterprise Java beans (EJB) applications. When you use EJB 3.1 components, you can create a distributed, secure application with transactional support. When you develop applications that access persistent data, you can use the Java Persistence API (JPA). This standard simplifies the creation and use of persistent entities, as well as adding new features. For developing presentation logic, you can use technologies such as JavaServer Pages (JSP) or JavaServer Faces (JSF).

- [Developing Java EE Applications](#)

You can develop Java™ applications with the Java EE programming model by using WebSphere Developer Tools. These topics provide instructions for creating, configuring, defining, and securing Java applications, and validating code within applications.

- [EJB 3.x overview](#)

You can use the workbench to develop and test enterprise beans that conform to the distributed component architecture defined in the Enterprise JavaBeans™ (EJB) 3.2 specification. EJB 3.2, 3.1, and 3.0 are supported.

- [Developing Java Persistence API \(JPA\) applications](#)

You can develop applications with the Java Persistence API (JPA), which is a simplification of the persistence programming model. These topics provide information on creating and configuring JPA projects and entity beans, and other features of the JPA tools.

- [Developing WebSocket applications](#)

WebSocket technology improves upon the existing HTTP request-response model, enables real-time communication, and improves performance and scalability for applications that require timely updates from the server. WebSocket applications consist of WebSocket endpoints, which can be client or server implementations of business logic.

- [Developing applications that use Contexts and Dependency Injection \(CDI\)](#)

You can use wizards to create applications that use Contexts and Dependency Injection (CDI).

Learn about enterprise applications

The workbench provides the tools you need to develop enterprise applications. You can use the Java EE tools and features to create applications that are structured around modules with different purposes, such as web sites and enterprise Java beans (EJB) applications. When you use EJB 3.1 components, you can create a distributed, secure application with transactional support. When you develop applications that access persistent data, you can use the Java Persistence API (JPA). This standard simplifies the creation and use of persistent entities, as well as adding new features. For developing presentation logic, you can use technologies such as JavaServer Pages (JSP) or JavaServer Faces (JSF).


Overview

You can read the following topics before creating an enterprise application. They provide planning and technology overview information that might be useful if you are new to enterprise applications or developing enterprise applications in this development environment.

-  [Overview of Java EE: Overview](#)
-  [Developing JPA applications](#)
-  [JPA architecture](#)

Getting started

If you are already familiar with enterprise applications technology the following topics help you set up your workspace for enterprise applications development, and guide you through the development process.

-  [Tools for Java EE development](#)
-  [Creating Java EE projects using wizards](#)
-  [Creating and configuring Java EE modules using annotations](#)
-  [Creating JPA applications](#)

Web resources for learning

In addition to the information found in this information center, the following links provide additional learning material.

[IBM® Redbooks®: Rational® Application Developer V7 Programming Guide](#)

[Java EE best practices](#)

[Latest developerWorks® articles and tutorials about enterprise applications](#)

[Latest developerWorks articles and tutorials about EJBs](#)

[Creating EJB projects](#)

[Developing EJB 3.1 Applications](#)

Parent topic: [Developing enterprise applications](#)

Developing Java EE Applications

You can develop Java™ applications with the Java EE programming model by using WebSphere Developer Tools. These topics provide instructions for creating, configuring, defining, and securing Java applications, and validating code within applications.

- [Learn about Java EE Applications](#)

The Java EE programming model simplifies the process of creating Java applications. You can deploy your Java EE application using WebSphere® Application Server V7.0 or later.

- [Java EE: Overview](#)

Using the Java Platform, Enterprise Edition (Java EE) architecture, you can build distributed web and enterprise applications. This architecture helps you focus on presentation and application issues, rather than on systems issues.

- [Selecting working sets](#)

You can select projects or components to group in working sets in the workspace.

- [Setting Java EE preferences](#)

Before you begin developing Java EE applications, you can optimize the workbench for Java enterprise development by setting various preferences.

- [Creating and configuring Java EE projects using wizards](#)

You can use wizards to create Java EE projects in your workspace.

- [Creating and configuring Java EE modules using annotations](#)

Annotations are a modifier or metadata tag that provide additional data to Java classes, interfaces, constructors, methods, fields, parameters, and local variables. With Java EE, you can use annotations to create and configure modules.

- [Defining Java EE applications](#)

You can use the tools in the workspace to define the module dependencies in the Java EE project in your workspace.

- [Enterprise application security](#)

You can provide security for your Java EE enterprise application using annotations or using deployment descriptors.

- [Setting permissions in the Java EE Application Permission Editor](#)

After you create your `permissions.xml` file, you can edit this file with the Java EE Application Permission Editor to set rights and permissions for your Java EE applications.

- [Validating code in enterprise applications](#)

The workbench includes validators that check certain files in your enterprise application module projects for errors.

Parent topic: [Developing enterprise applications](#)

Related concepts:

[Learn about Java EE Applications](#)

[Java EE: Overview](#)

[Tools for Java EE development](#)

[Project facets](#)

[Java EE with annotations overview](#)

[Defining Java EE applications](#)

[Enterprise application security](#)

Learn about Java EE Applications

The Java™ EE programming model simplifies the process of creating Java applications. You can deploy your Java EE application using WebSphere® Application Server V7.0 or later.

Java Enterprise applications (Java EE applications) are applications that conform to the Java Platform, Enterprise Edition (Java EE) specification. Before Java EE, the specification name was Java 2 Platform, Enterprise Edition (J2EE). The term Java EE includes Java EE and J2EE specifications.

In the Java EE specifications, programming requirements have been streamlined, and XML deployment descriptors are optional. Instead, you can specify many details of assembly and deployment with Java annotations. Java EE provides default values in many situations so that explicit specification of these values is not required.

Code validation, content assistance, Quick Fixes, and refactoring simplify working with your code. Code validators check your projects for errors. When an error is found, you can double-click it in the Problems view in the product workbench to go to the error location. For some error types, you can use a Quick Fix to correct the error automatically. For both Java source and Java annotations, you can rely on content assistance to simplify your programming task. When you refactor source code, the tools automatically update the associated metadata.

For additional information about Java EE, see the official specification:

- **Java EE 5:** [JSR 244: Java Platform, Enterprise Edition 5 \(Java EE 5\) Specification](#)
- **Java EE 6:** [JSR 316: Java™ Platform, Enterprise Edition 6 \(Java EE 6\) Specification](#)

Parent topic: [Developing Java EE Applications](#)

Related concepts:

- [Java EE: Overview](#)
- [Tools for Java EE development](#)
- [Project facets](#)
- [Java EE with annotations overview](#)
- [Defining Java EE applications](#)
- [Enterprise application security](#)

Related tasks:

- [Developing Java EE Applications](#)
- [Setting Java EE preferences](#)
- [Creating and configuring Java EE projects using wizards](#)
- [Validating code in enterprise applications](#)

Java EE: Overview

Using the Java™ Platform, Enterprise Edition (Java EE) architecture, you can build distributed web and enterprise applications. This architecture helps you focus on presentation and application issues, rather than on systems issues. You can use the Java EE tools and features to create applications that are structured around modules with different purposes, such as web sites and Enterprise Java bean (EJB) applications. When you use EJB 3.1 components, you can create a distributed, secure application with transactional support. When you develop applications that access persistent data, you can use the Java Persistence API (JPA). This standard simplifies the creation and use of persistent entities. For developing presentation logic, you can use technologies such as JavaServer Pages (JSP) or JavaServer Faces (JSF). Using the Java EE Platform Enterprise Edition (Java EE), you can develop applications more quickly and conveniently than in previous versions. Java EE significantly enhances ease of use providing:

- Reduced development time
- Reduced application complexity
- Improved application performance

Java EE provides a simplified programming model, including the following tools:

- Inline configuration with annotations, making deployment descriptors now optional
- Dependency injection, hiding resource creation, and lookup from application code
- Java persistence API (JPA) allows data management without explicit SQL or JDBC
- Use of plain old Java objects (POJOs) for Enterprise Java beans and web services

Java EE provides simplified packaging rules for enterprise applications:

- Web applications use .WAR files
- Resource adapters use RAR files
- Enterprise applications use .EAR files
- The `lib` directory contains shared .JAR files
- A .JAR file can be specified in the `application.xml` of the EAR either as an application client or as an EJB module
- A .JAR file that is not specified by the `application.xml` of the EAR is defined as follows:
 - A .JAR file with an `application-client.xml` implies an application client
 - A .JAR file with an `ejb-jar.xml` implies an EJB module
 - A .JAR file with `META-INF/MANIFEST.MF` specified `Main-Class` implies an application client
 - A .JAR file with any `@Stateless`, `@Stateful`, or `@MessageDriven` annotations implies an EJB application
- A .JAR file with `Main-Class` implies an application client
- A .JAR file with `@Stateless` annotation implies an EJB application
- Many simple applications no longer require deployment descriptors, including
 - EJB applications (.JAR files)
 - Web applications that use JSP technology only
 - Application clients
 - Enterprise applications (.EAR files)

Java EE provides simplified resource access using dependency injection:

- In the Dependency Injection pattern, an external entity automatically supplies an object's dependencies.
 - The object does not need to request these resources explicitly
- In Java EE, dependency injection can be applied to all resources that a component needs
 - Creation and lookup of resources are hidden from application code
- Dependency injection can be applied throughout Java EE technology:
 - EJB containers

- Web containers
- Clients
- Web services

- **Support for Java EE 7**

Java EE 7 provides a number of new or enhanced features, which this release supports.

- **Tools for Java EE development**

The Java EE development tools enable you to create enterprise applications including enterprise application projects, EJBs, and Java persistent API (JPA) applications.

- **Project facets**

Facets define characteristics and requirements for Java EE projects and are used as part of the runtime configuration.

Parent topic: [Developing Java EE Applications](#)

Related concepts:

[Learn about Java EE Applications](#)

[Tools for Java EE development](#)

[Project facets](#)

[Java EE with annotations overview](#)

[Defining Java EE applications](#)

[Enterprise application security](#)

Related tasks:

[Developing Java EE Applications](#)

[Setting Java EE preferences](#)

[Creating and configuring Java EE projects using wizards](#)

[Validating code in enterprise applications](#)

Support for Java EE 7

Java™ EE 7 provides a number of new or enhanced features, which this release supports.

WebSphere Developer Tools now supports the Java™ Platform, Enterprise Edition (Java EE) 7. The new and enhanced features of this platform include HTML5 support, improved support for batch applications, method-level validation through Bean Validation 1.1, and many other improvements.

See [Java EE 7 in Liberty profile](#) for more information about Liberty profile support for Java EE 7.

See [JSR 342: Java™ Platform, Enterprise Edition 7 \(Java EE 7\) Specification](#) for full details about Java EE 7.

Parent topic: [Java EE: Overview](#)

Tools for Java EE development

The Java™ EE development tools enable you to create enterprise applications including enterprise application projects, EJBs, and Java persistent API (JPA) applications.

The workbench provides a rich set of tools to create, develop, test, debug, and deploy enterprise applications. Some of the Java EE tools include:

- **Enterprise explorer view:** The Enterprise explorer view allows you to manage and maintain your enterprise applications in one location.
- **Annotations view:** The Annotations view provides a way for you to create, edit, browse, and generally keep track of the annotations that you use in your applications.
- **Project and object creation wizards:** You can use Java EE wizards to create enterprise applications, including application client projects, Enterprise Java bean projects, connector projects, web projects, web fragment projects, and web Services applications.
- **Export and import wizards:** You can import existing projects and modules into your current project; you can use export wizards to build, package, and export projects with a single click.
- **Code validation, content assistance, Quick Fixes, and refactoring:** These tools simplify working with your code. Code validators check your projects for errors. When one is found, you can double-click it, in the Problems view in the product workbench, to go to the error location. For some types of error, you can also choose a Quick Fix, which automatically corrects the error. For both Java™ source and Java annotations, you can rely on content assistance to simplify your programming task. When you refactor source code, the tools automatically update the associated metadata.
- **Tools for managing projects:** The tools for Java EE allow you to manage projects easily. To share applications outside of a version-control system, you can import and export projects as archive files.

- Java EE perspective

While developing enterprise applications in the Java EE perspective, the enterprise explorer view is your main view of your Java EE projects and resources.

- Enterprise explorer view in the Java EE perspective

While developing enterprise applications in the Java™ EE perspective, the enterprise explorer view is the main view of your Java EE projects and resources.

Parent topic: [Java EE: Overview](#)

Related concepts:

[Learn about Java EE Applications](#)

[Java EE: Overview](#)

[Project facets](#)

[Java EE with annotations overview](#)

[Defining Java EE applications](#)

[Enterprise application security](#)

[Enterprise explorer view in the Java EE perspective](#)

[Java EE perspective](#)

Related tasks:

[Developing Java EE Applications](#)

[Setting Java EE preferences](#)

[Creating and configuring Java EE projects using wizards](#)

[Validating code in enterprise applications](#)

Selecting working sets

Java EE perspective

While developing enterprise applications in the Java™ EE perspective, the enterprise explorer view is your main view of your Java EE projects and resources.

The Java EE perspective contains a selection of views and editors that are customized to be most useful to an Enterprise developer.

Views

The Java EE perspective includes workbench views that you can use when developing resources for enterprise applications, EJB modules, web modules, web fragment modules, application client modules, and connector projects or modules:

- **Enterprise explorer view:** The Enterprise Explorer view provides an integrated view of your projects and their artifacts related to Java EE development. You can show or hide your projects based on working sets. This view displays navigable models of Java EE deployment descriptors, Java artifacts (source folders, packages, and classes), navigable models of the available web services, and specialized views of web modules to simplify the development of web applications. In addition, EJB database mapping and the configuration of projects for a Java EE application server are made readily available.
- **Annotations view:** The Annotations view provides a way for you to create, edit, browse, and generally keep track of the annotations that you use in your applications.
- **Outline view:** The Outline view in the Java EE perspective shows the outline of the file that you are editing. For example, if you are editing an enterprise bean in the Java editor, the Outline view shows the outline for the Java class.
- **Tasks view:** The Tasks view lists the to-do items that you have entered.
- **Problems view:** The Problems view displays problems, warnings, or errors associated with the selected project. You can double-click an item to address the specific problem in the appropriate resource.
- **Properties view:** The Properties view provides a tabular view of the properties and associated values of objects in files you have open in an editor.
- **Servers view:** The Servers view shows all the created server instances. You can start and stop each server from this view, and you can launch the test client.
- **Snippets view:** The Snippets view provides categorized pieces of code that you can insert into appropriate places in your source code.
- **Data Source Explorer:** The Data Source Explorer provides a list of configured connection profiles. If categories are enabled, you can see the list grouped into categories. Use the Data Source Explorer to connect to, navigate, and interact with resources associated with the selected connection profile. It also provides import and export capabilities to share connection profile definitions with other Eclipse Workbenches.
- **Status bar:** The Status bar provides a description of the location of selected objects in the Enterprise Explorer view. When file and deployment descriptors are open, the status bar shows the read-only state of the files and the line and column numbers when applicable. Sometimes when long operations run, a status monitor appears in the status bar, along with a button with a stop sign icon. Clicking the stop sign stops the operation when the operation can be canceled.

Editors

Editors are workplace tools that allow you to edit the various types of files that are contained in your project. Depending on the type of file that is being edited, the appropriate editor is displayed in the editor area. For example, if a .TXT file is being edited, a text editor is displayed in the editor area. The following list contains some of the editors available to you in the Java EE development environment:

- **Deployment Descriptor Editors:** The editor includes editors for managing your various Java EE projects types.
- **Java Editor:** The editor includes the following features:
 - Syntax highlighting

- Content and code assist
- Code formatting
- Import assistance
- Quick fix
- Integrated debugging features
- **Ant editor:** The editor includes the following features:
 - Syntax highlighting
 - Content and code assist (including Ant-specific templates)
 - Annotations
- **XML editor:** The XML editor is a tool for creating and viewing XML files. You can use it to perform various tasks, such as:
 - Creating new, empty XML files, or generating them from existing DTDs or existing XML schemas
 - Editing XML files
 - Importing existing XML files for structured viewing
 - Associating XML files with DTDs or XML schemas
- **Properties files editor:** The Properties files editor is a tool for creating and viewing Properties files. The editor includes the following features:
 - Syntax highlighting
 - Code formatting

Parent topic: [Tools for Java EE development](#)

Related concepts:

[Tools for Java EE development](#)

[Enterprise explorer view in the Java EE perspective](#)

Related tasks:

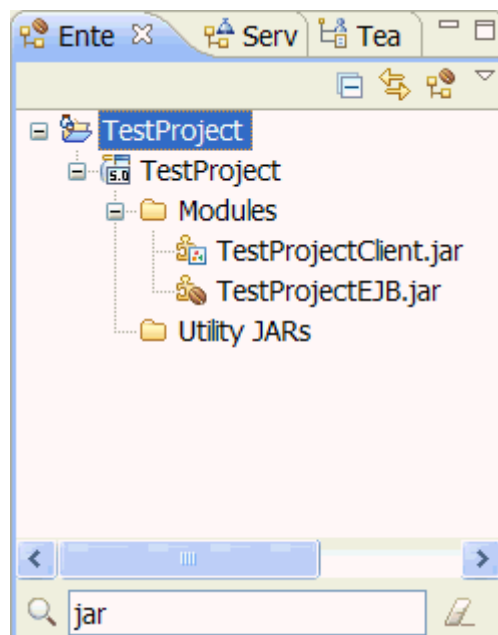
[Selecting working sets](#)

Enterprise explorer view in the Java EE perspective

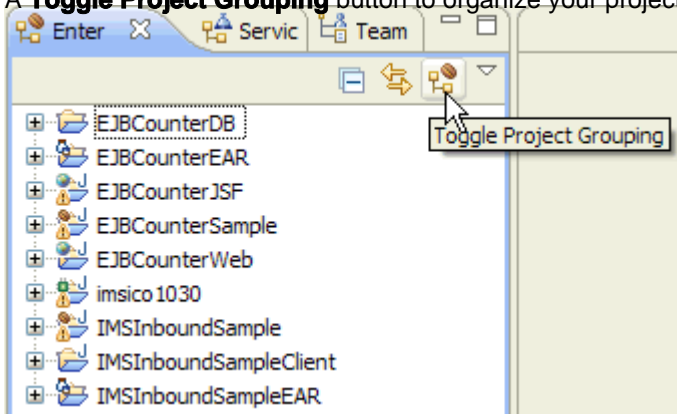
While developing enterprise applications in the Java™ EE perspective, the enterprise explorer view is the main view of your Java™ EE projects and resources.

The enterprise explorer view in the Java EE perspective is a view that allows you to manage and maintain your enterprise applications in one location. The enterprise explorer view provides you with

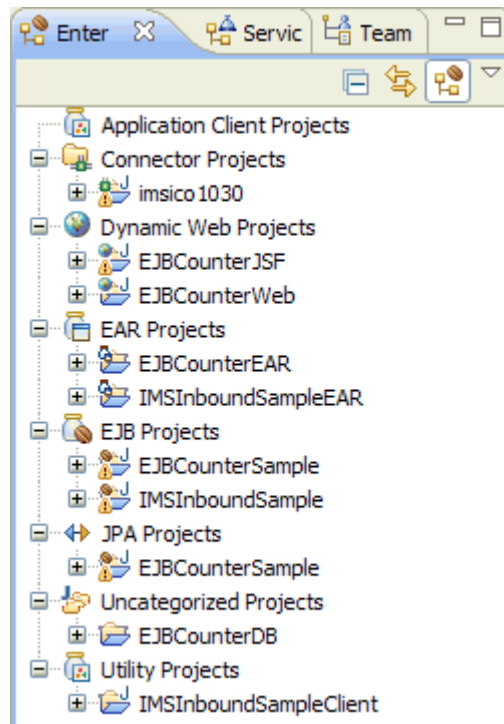
- An integrated view of all project resources, including models of Java EE deployment descriptors (if you are using deployment descriptors), Java artifacts, resources, web services, databases, and web project artifacts.
- A way to view all the modules associated with your project.
- Search bar capabilities that allow you to filter the files in your project folders according to the search parameter you type in the search bar. For example, if you type `xml` in the search bar and click **Search**, only the xml files show up in the Enterprise Explorer. If you type `jar` in the search bar and click **Search**, only the JAR files in your project (excluding the jars on the classpath) show up:



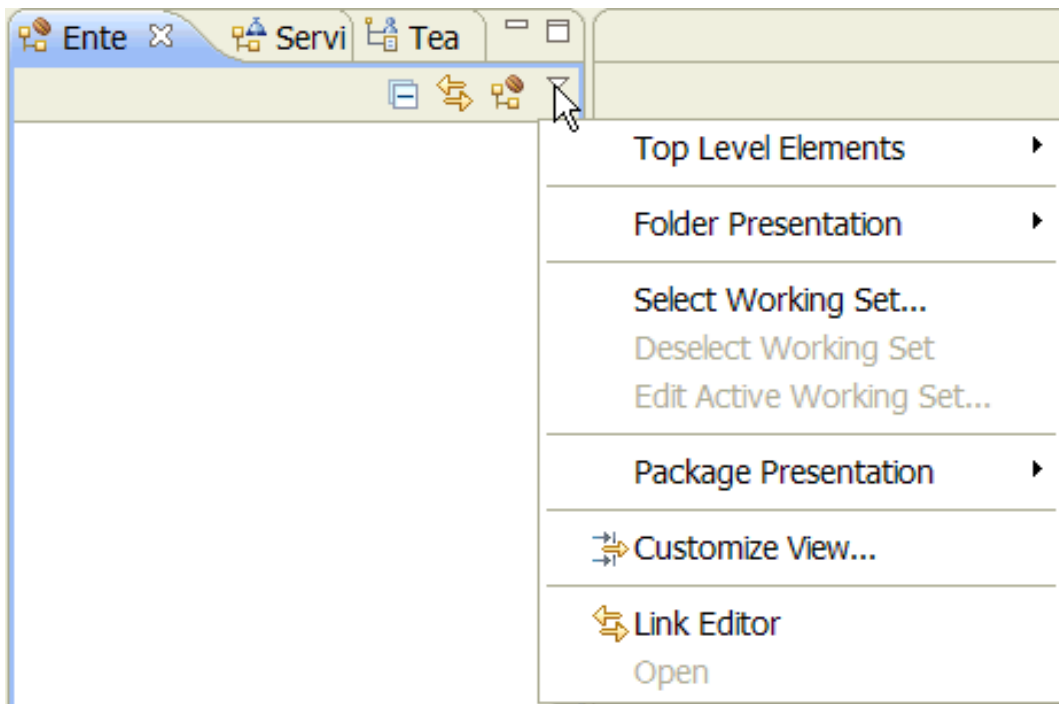
- A **Toggle Project Grouping** button to organize your projects by project type:



The result is a tree organized by project type:



- Customization options, so that you can select what functionality you want available in the enterprise explorer view. Click the **View Menu** arrow in the Enterprise Explorer view, and select **Customize View...**:



- Version control management information (VCM) can be turned on and off from the Preferences page: **Window > Preferences > General > Appearance > Label decorations.**
- Classpath module dependencies can be modified by selecting **Window > Preferences > General > Appearance > Label decorations**
- View filtering is supported by clicking the **View Menu** arrow in the Enterprise Explorer view, selecting **Customize View... > Filters.** Resources can be filtered by name, project type, or content type. Files beginning with a period are filtered out by default.
- The status line shows the full path of the selected resource.

- Errors and warnings on resources (including Java, HTML/JSP, and Links Builder errors and warnings) are indicated with a red error or yellow warning next to the resource with the error, as well as the parent containers up to the project.

Parent topic: [Tools for Java EE development](#)

Related concepts:

[Tools for Java EE development](#)

[Java EE perspective](#)

Related tasks:

[Selecting working sets](#)

Project facets

Facets define characteristics and requirements for Java™ EE projects and are used as part of the runtime configuration.

When you add a facet to a project, that project is configured to perform a certain task, fulfill certain requirements, or have certain characteristics. For example, the EAR facet sets up a project to function as an enterprise application by adding a deployment descriptor and setting up the project's class path.

You can add facets to Java EE projects and other types of projects that are based on Java EE projects, such as enterprise application projects, web projects, and EJB projects. You can add facets to a Java project, by selecting the properties page of the project and selecting **Project Facets > Convert to faceted form....** Typically, a facet-enabled project has at least one facet when it is created, allowing you to add more facets if necessary. For example, a new EJB project has the EJB Module facet. You can then add other facets to this project like the EJBDoclet (XDoclet) facet. To add a facet to a project, see [Adding a facet to a Java EE project](#).

Some facets require other facets as prerequisites. Other facets cannot be in the same project together. For example, you cannot add the Web Module facet to an EJB project because the EJB project already has the EJB Module facet. Some facets can be removed from a project and others cannot.

Facets also have version numbers. You can change the version numbers of facets as long as you stay within the requirements for the facets. To change the version number of a facet, see [Changing the version of a facet](#).

Parent topic: [Java EE: Overview](#)

Related tasks:

[Changing the Java compiler version for a Java EE project](#)

Selecting working sets

You can select projects or components to group in working sets in the workspace.


Before you begin

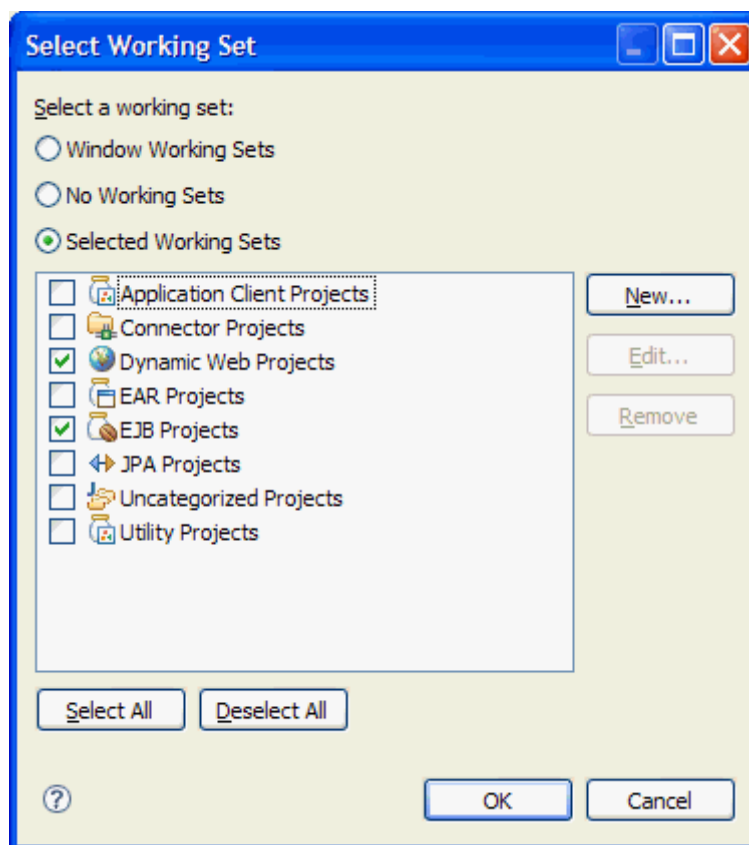
You must have a project or several projects created within your workspace.

About this task

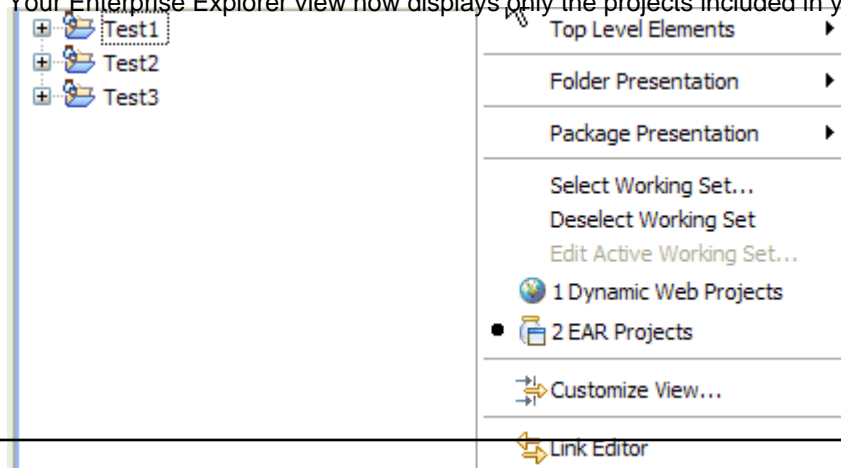
Working sets group elements for display in views or for operations on a set of elements. The navigation views use working sets to restrict the set of resources that are displayed. If a working set is selected in the navigator, only resources, children of resources, and parents of resources that are contained in the working set are shown.

Procedure

1. In the Enterprise Explorer view of the Java™ EE perspective, click the **View** Menu icon, , and select **Select working Set**.
2. Select the type of project or projects that you want to group in your working set:



3. Your Enterprise Explorer view now displays only the projects included in your working set:



4. You can create more than one working set, and you can toggle back and forth between a number of working sets in your workspace.

What to do next

For more information about working sets, see [Working sets](#)

Parent topic: [Developing Java EE Applications](#)

Related concepts:

[Tools for Java EE development](#)

[Enterprise explorer view in the Java EE perspective](#)

[Java EE perspective](#)

Setting Java EE preferences

Before you begin developing Java™ EE applications, you can optimize the workbench for Java enterprise development by setting various preferences.

Procedure

1. Click **Window** > **Preferences** to open the Preferences page.
2. Expand Java EE, and click the preference category that you want to set.
3. Select the elements that you want to set as the default when creating your Java EE application. These Java EE settings include: JET templates:
 - Use dynamic translation of JET templates (default: cleared)

Deployment:

- Perform Incremental Deployment (default: checked)

Classpath containers:

- Use Ear Libraries classpath container (default: checked)
 - Use Java Build Path to control EAR Libraries export (default: Cleared)
- Use Web App Libraries classpath container (default: checked)

Classpath checking:

- Duplicate Java EE Module dependency classpath entries:
 - Error
 - Warning (default)
 - Ignore
- Duplicate Web Library dependency classpath entries:
 - Error
 - Warning (default)
 - Ignore
- Missing Java EE Modules dependency classpath entries:
 - Error
 - Warning (default)
 - Ignore
- Missing Web Library dependency classpath entries:
 - Error
 - Warning (default)
 - Ignore

4. Click **Apply** to apply the changes and click **OK** to close the Preferences page.

What to do next

For information about other Java EE preferences, refer to [Setting Java compiler compliance](#) and the following subtopic:

- [Selecting default project structures](#)

You can use the Preferences page to set your default project structures.

Parent topic: [Developing Java EE Applications](#)

Related concepts:

[Learn about Java EE Applications](#)

[Java EE: Overview](#)

[Tools for Java EE development](#)

[Project facets](#)

[Java EE with annotations overview](#)

[Defining Java EE applications](#)

[Enterprise application security](#)

Selecting default project structures

You can use the Preferences page to set your default project structures.

Procedure

1. Click **Window > Preferences** to open the Preferences page.
2. Expand Java™ EE, and select **Project**
3. Select the elements that you want to set as the default when creating your Java EE application. These project settings include: Enterprise Application Membership:
 - Add project to an EAR (default: checked)

Content directory (indicates the default directory in your workspace where the indicated project is stored. You can change the name of the default directory here):

- Enterprise Application Project:
 - Content Directory: (default: unspecified)
- Web Project:
 - Default Source Folder: (default: src)
 - Output Folder: (default: WebContent/WEB-INF/classes)
 - Content Directory: (default: WebContent)
- EJB Project:
 - Default Source Folder: (default: ejbModule)
 - Output Folder: (default: build\classes)
- Application Client Project:
 - Default Source Folder: (default: appClientModule)
 - Output Folder: (default: build\classes)
- Connector Application Project:
 - Default Source Folder: (default: connectorModule)
 - Output Folder: (default: build\classes)
- Utility/JPA Project:
 - Default source folder: (default: src)
 - Output Folder: (default: src)

Generate Deployment Descriptor for Java EE 5.0 or later Projects (Select this option if you want a deployment descriptor to be automatically created when you create the project type indicated):

- Enterprise Application Project (default: Cleared)
- EJB Project (default: Cleared)
- Web 2.5 Project (default: checked)
- Web 3.0 Project (default: cleared)
- Application Client Project (default: cleared)
- Connector Project (applies for 1.6 only) (default: checked)

4. Click **Apply** to apply the changes and click **OK** to close the Preferences page.

- **Setting Java compiler compliance**

Use the Workspace Preferences page to set the Java compiler compliance.

Parent topic: [Setting Java EE preferences](#)

Related tasks:

[Setting Java compiler compliance](#)

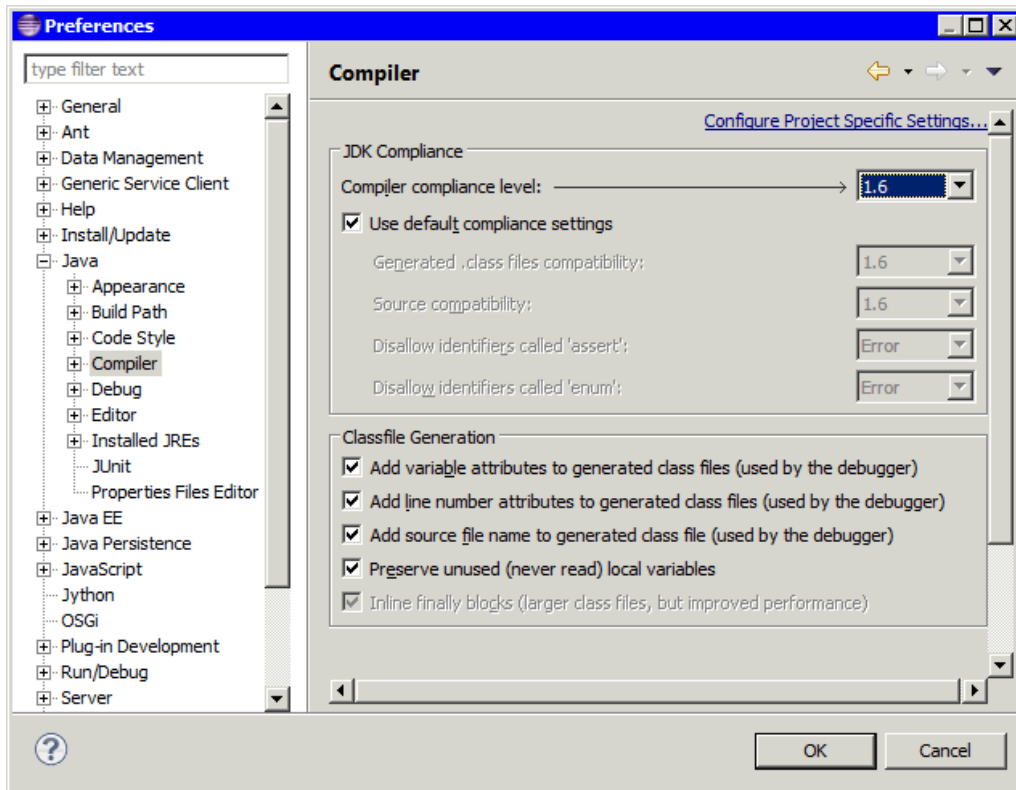
[Setting Maven preferences](#)

Setting Java compiler compliance

Use the Workspace Preferences page to set the Java™ compiler compliance.

Procedure

1. To open the workspace preferences page, select **Window > Preferences > Java > Compiler**. The Compiler page opens:



2. In the **Compiler compliance level** field, select the compiler level that you want to use for your application. You must select the compiler level that is supported by the version of the WebSphere® Application Server to which you are deploying.
 - A. If you are using WebSphere Application Server version 8.5 or later, select Java 1.6 or Java 1.7 compiler compliance.
 - B. If you are using WebSphere Application Server version 7.0 or 8.0, select Java 1.6 compiler compliance.
 - C. If you are using WebSphere Application Server version 7.0, select Java 1.6 compiler compliance. For more information about compiler versions that are supported by WebSphere Application Server, see [Testing and publishing on a server](#).

Parent topic: [Selecting default project structures](#)


Related tasks:

[Selecting default project structures](#)

Creating and configuring Java EE projects using wizards

You can use wizards to create Java™ EE projects in your workspace.

Procedure

1. If you do not see the Java EE icon,  , in the workspace, [switch to the Java EE perspective](#).
2. Create and configure your Java EE project. Subtopics describe how to create, configure, delete, and import Java EE projects or modules.

- [Switching to Java EE Perspective](#)

To use the Java EE dynamic wizards to create applications, set your workspace to the Java EE perspective.

- [Creating enterprise application projects](#)

You can use wizards to create an enterprise application project in your Java EE project.

- [Creating application client modules](#)

You can use wizards to create an application client module in your Java EE project.

- [Creating EJB modules using wizards](#)

You can use wizards to create an EJB module in your Java EE project.

- [Creating web modules](#)

You can use wizards to create web modules in your Java EE project.

- [Creating web fragment projects](#)

You can use the web fragment project wizard to create web fragment projects in your workspace.

- [Creating connector modules](#)

You can use wizards to create a connector module in your Java EE project.

- [Deleting Java EE modules](#)

You can delete modules from existing Java EE applications.

- [Importing JAR, WAR, EJB JAR, client application JAR, and RAR files](#)

- [Exporting JAR, WAR, EJB JAR, client application JAR, and RAR files](#)

- [Customizing the Liberty class path container](#)

You can customize the contents of the Liberty profile class path container by excluding third-party API libraries, IBM® API libraries, and unrecognized content in your Java EE project.

- [Configuring Bean Validation](#)

You can use Bean Validation to use one or more constraints to verify data that is used in application logic. You can use predefined constraints, or you can create your own.

Parent topic: [Developing Java EE Applications](#)

Related concepts:

[Learn about Java EE Applications](#)

[Java EE: Overview](#)

[Tools for Java EE development](#)

[Project facets](#)

[Java EE with annotations overview](#)

[Defining Java EE applications](#)

[Enterprise application security](#)

Creating enterprise application projects

You can use wizards to create an enterprise application project in your Java™ EE project.

Before you begin

If you do not see the Java EE icon,  in the workspace, you can [switch to the Java EE perspective](#).

Procedure


1. To use the Java EE wizard from the menu bar, select **File > New > Project...Java EE > Enterprise Application Project**. Alternatively, right-click the Enterprise Explorer view, and select **New > Enterprise Application Project**.
2. On the Enterprise Application project page, provide the following information:
 - In the **Project name** field, type a name for your Enterprise Application project.
 - In the **Project location** field, accept the default, or clear **Use default location** and type a directory name to contain your project contents.
 - In the **Target Runtime** field, select a target run time from the drop-down list or click **New Runtime** to create an application server run time.
 - In the **EAR version** field, accept the default or select a different version from the drop-down list.
 - In the **Configuration** field, select the wanted application server configuration, or click **Modify** to modify the configuration.
3. Click **Next**.
4. On the Configure enterprise application settings page, provide the following information:
 - In the **Java EE Module dependencies** field, select a module or click **New Module** to create a module.
 - Type a directory location name in the **Content Directory:** field.
 - Accept the default value in the **Generate deployment descriptor** field (cleared), or select if you want to generate a deployment descriptor.
5. Click **Finish**.

Parent topic: [Creating and configuring Java EE projects using wizards](#)

Creating application client modules

You can use wizards to create an application client module in your Java™ EE project.

Before you begin

If you do not see the Java EE icon, , you can [switch to the Java EE perspective](#).

Procedure

1. To use the Java EE wizards from the menu bar, select **File > New > Application Client Project**.
2. Alternatively, right-click the Enterprise Explorer view, and select **New > Other > Java EE > Application Client Project**.
3. On the Application Client module page, provide the following information:
 - In the **Project name** field, type a name for your Application Client project.
 - In the **Project location** field, accept the default, or clear **Use default location** and type a directory name to contain your project contents.
 - In the **Target Runtime** field, select a target run time from the drop-down list or click **New Runtime** to create an application server run time.
 - In the **Application Client module version** field, accept the default or select a different version from the drop-down list.
 - In the **Configuration** field, select the wanted application server configuration, or click **Modify** to modify the configuration.
 - In the **EAR Membership** field, accept the default **Add project to an EAR** and accept the default EAR project name, or type an alternative in the **EAR Project Name** field. To create an EAR project, click **New Project**.
4. Click **Next**.
5. On the Configure Application Client module settings page, provide the following information:
 - In the **Source Folder** field, specify a folder for your source files by clicking the **Add Folder** button or accept the default value (appClientModule).
 - In the **Output Folder:** field, specify a folder for your output files or accept the default value `build\classes`.
6. Click **Next**.
7. On the Application Client module page, provide the following information:
 - In the **Create a default Main class** field, accept the default (selected) or clear if you do not want to create a main class.
 - In the **Generate deployment descriptor** field, accept the default (cleared), select if you want to generate a deployment descriptor.
8. Click **Finish**.

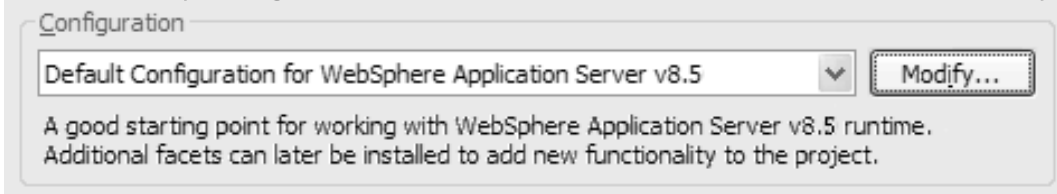
Parent topic: [Creating and configuring Java EE projects using wizards](#)

Creating EJB modules using wizards

You can use wizards to create an EJB module in your Java™ EE project.

Procedure

1. In the Java EE perspective, right-click your enterprise application project and select **New > EJB Project**. The New EJB Project wizard opens.
2. In the **Name** field, type a name for the EJB project. To change the default Project location, clear the **Use default location** checkbox and click the **Browse** button to select a new location.
3. In the **Target runtime** drop-down list, select the server that you want to target for your development. Or, create a target runtime environment by clicking **New**. The target runtime selection affects the compilation and runtime settings by modifying the class path entries for the project. To create an EJB 3.1 project, select the appropriate version of IBM® WebSphere® Application Server.
4. Select the EJB module version or accept the default.
5. Optional: Select a pre-defined project configuration from the **Configurations** drop-down list.
6. Optional: **Modify Configuration**: If you want to modify the configuration details, click **modify**:



If you are creating an entity, for example, select **Java Persistence**. Save this configuration with a meaningful name, for example, `EJBDevelopmentWithEntityBeans` so that you can reference this configuration in any EJB 3.1 projects that are subsequently created.

7. Optional: Select **Add project to an EAR module** to add the new module to an enterprise module (EAR) project. Type a new project name or select an existing enterprise module project from the drop-down list in the **EAR Project Name** combination box, or, click **New Project** to launch the New EAR module Project wizard.
8. Click **Next**.
9. In the **Source Folder** field, specify a folder for your source files by clicking the **Add Folder...** button or accept the default value (`appClientModule`).
10. In the **Output Folder** field, specify a folder for your output files or accept the default value (`build\classes`).
11. Optional: If in the previous page, you selected Add project to an EAR module, then you are able to create an EJB Client JAR. Select **Create an EJB Client JAR module** to hold the client interfaces and classes if you want the client interface and classes for your enterprise beans to be kept in a separate EJB client JAR file. This EJB client JAR file is added to the enterprise module as a project utility JAR file. Specify values for the **Name** and **Client JAR URI** fields, or accept the defaults. If you select this option, the Deployment Descriptor is generated by default.
12. If you cleared the **Create an EJB Client JAR module** to hold the client interfaces and classes checkbox, you can select **Generate Deployment Descriptor** if you want to create a deployment descriptor, although the deployment descriptor is optional in EJB 3.0 and later. The deployment descriptor stores information relating to the EJB project in an Extensible Markup Language (XML) file, serving three functions:
 - Declaring the contents of the module
 - Defining the structure and external dependencies of the beans in the module
 - Describing how the enterprise beans are to be used at run timeYou can also add a deployment descriptor to your EJB module later. See [Generating deployment descriptors](#).
13. Click **Finish**.

Parent topic: [Creating and configuring Java EE projects using wizards](#)

Creating web modules

You can use wizards to create web modules in your Java™ EE project.

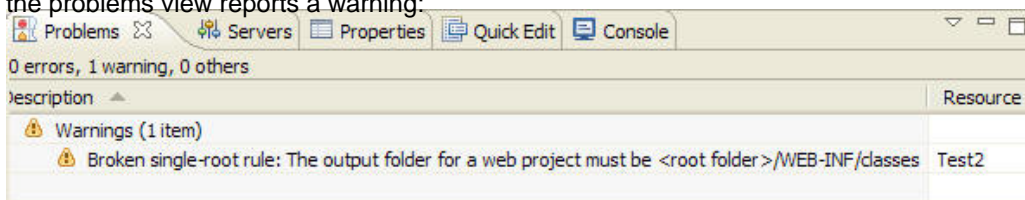
Procedure

1. In the Java EE perspective, right-click your enterprise application project and select **New > Web Project** to open the web project wizard.
2. In the **Name** field, type a name for your new web project.
3. In the Project Templates section, select the type of web template you want to use:

Option	Description
Dojo Toolkit	Configures the project to have Dojo capabilities. The Dojo resources can be in the project itself, a separate project, or a remote location accessible via HTTP.
REST Services	A project that is configured for REST Services based on JAX-RS
Simple	This creates a basic web project.
JQuery	JQuery Configures the project to have JQuery capabilities. The JQuery resources should be copied in the project itself.

4. In the Programming Model section, select the programming model that you want to use:
 - Client-side only (HTML, JavaScript,...)
 - Java EE
 - OSGiClick **Next** to configure your new web project.
5. On the deployment page, from the list of available configuration options, click **Deployment** to open the Deployment configuration page.
 - The **Target runtime** field is pre-populated with the selection from the enterprise project. You can change **Target runtime** by selecting another one from the drop-down box. Click **Change Features** to open the Project Facets window.
 - Click **Add support for WebSphere bindings and extensions** or clear this field.
 - In the **Web module version** field, select the web module version that you want to use.
 - In the **EAR membership** field, click **Add project to an EAR**, if you want to include EAR membership; clear this field if you do not want to add the web project to an EAR file.
 - In the **EAR project name** field, the name of your existent EAR file appears. You can click **Browse** to select a different EAR file.

Note: The deployment option is not available if you selected the Client-side only programming model for your new web project.
6. From the list of available configuration options, click **Java** to open the Java configuration page.
 - In the **Source folders on build path** field, accept the default **src** directory, or click **Add Folder**, **Edit...** or **Remove** to specify a folder for your source files.
 - In the **Default output folder:** field, specify a folder for your output files or accept the default value (WebContent\WEB-INF\classes). **Important:** If you choose a folder other than WebContent\WEB-INF\classes for your default output folder, the problems view reports a warning:



The default for single rootedness problems is set to warning. To change this setting, use the Validation Filters for Project Structure Validator page:

- A. Click **Window > Preferences > Validation > Project Structure Validation** and then the **...** button for **Settings**.
- B. In the Validation Filters for Project Structure Validator page, specify the default severity level. Available severity levels are **Error**, **Warning**, and **Ignore**.

7. From the list of available configuration options, click **Web Module**. On the Web Module configuration page:
 - In the **Context root** field, type the name of your web project root, or accept the default (which is the name of your web project).
 - In the **Content directory** field, type the name of your content directory, or accept the default (WebContent).
 - Select **Generate web.xml deployment descriptor** if you want to create a deployment descriptor. You can also add a deployment descriptor to your web module later.
8. Click **Finish** to create your web project.

- **Using loose class path web libraries support**

The workbench provides loose class path web libraries support for your web projects.

- **Creating and configuring resource references for Web 2.5, Web 3.0, and Web 3.1**

You can create and configure resource references for Web 2.5, Web 3.0, and Web 3.1 projects using the deployment descriptor.

Parent topic: [Creating and configuring Java EE projects using wizards](#)

Using loose class path web libraries support

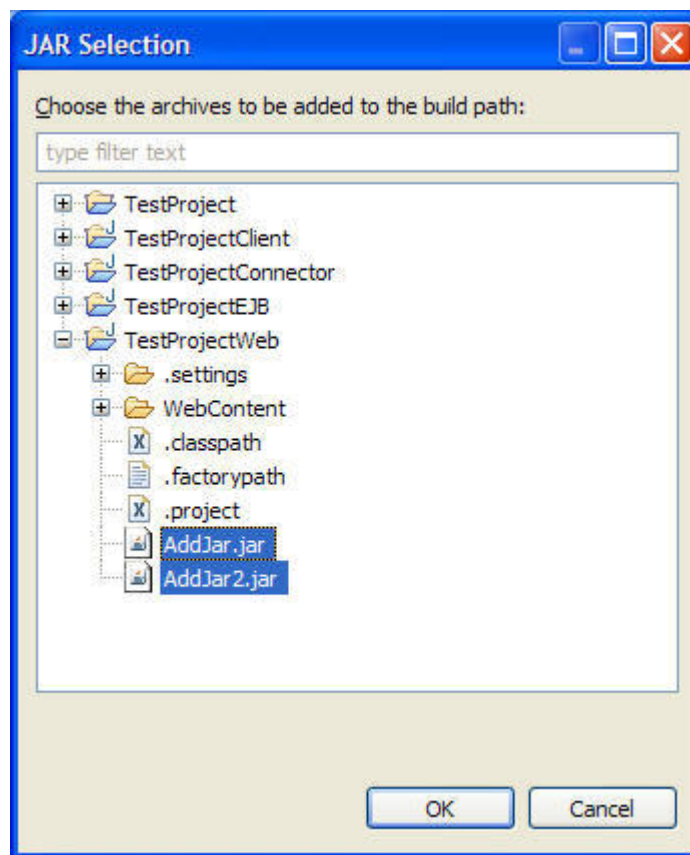
The workbench provides loose class path web libraries support for your web projects.

About this task

Loose class path support is an optional mechanism that allows for the inclusion of Eclipse Java™ development tools class path artifacts in a web project's WEB-INF/lib folder. You can add dependences using the Deployment Assembly page, but you can also add the web libraries from the Libraries page under the Java Build Page. You can include JAR files, external JAR files, libraries, and variables. The project validator detects loose class path issues and reports them in the problems view.

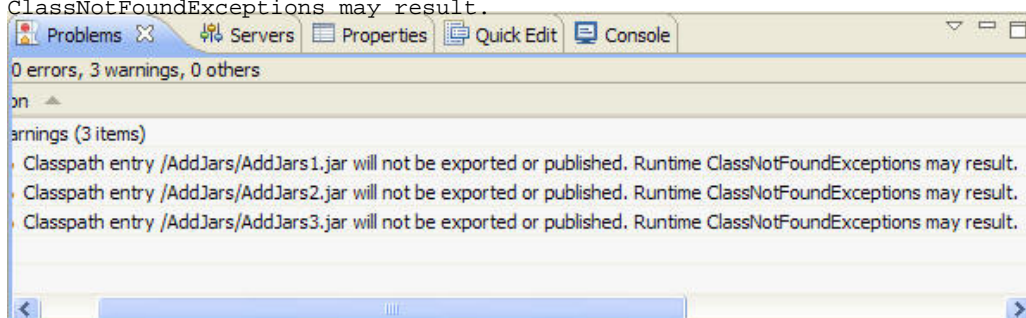
Procedure

1. Right-click your web project and select **Properties > Java Build Path > Libraries**.
2. Select **Add Jars**, and select the JAR files that you want to add to your web project. Click **OK**.

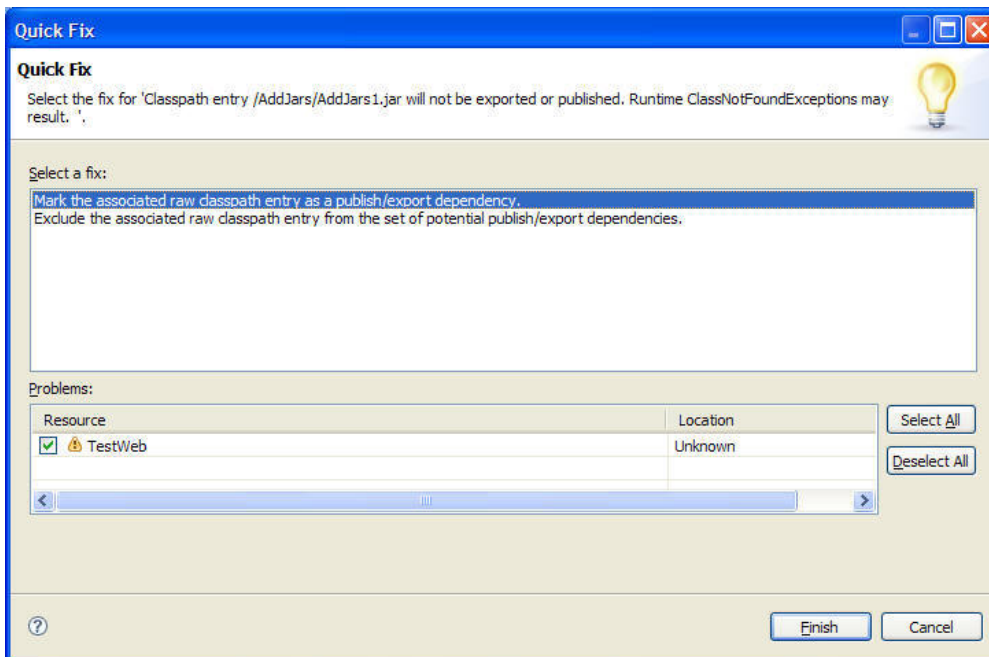


3. Because these JAR files are added only to the build path and are linked as components, if you deploy this EAR/WAR or export this EAR/WAR none of these JAR files are included. A warning message appears in the Problems view:

Classpath entry /AddJars/AddJars1.jar will not be exported or published. Runtime
ClassNotFoundExceptions may result.



4. Right-click the warning message and select **Quick Fix**. You can either:
 - Mark the associated raw class path entry as a publish/export dependency.
 - Exclude the associated raw class path entry from the set of potential publish/export dependencies.



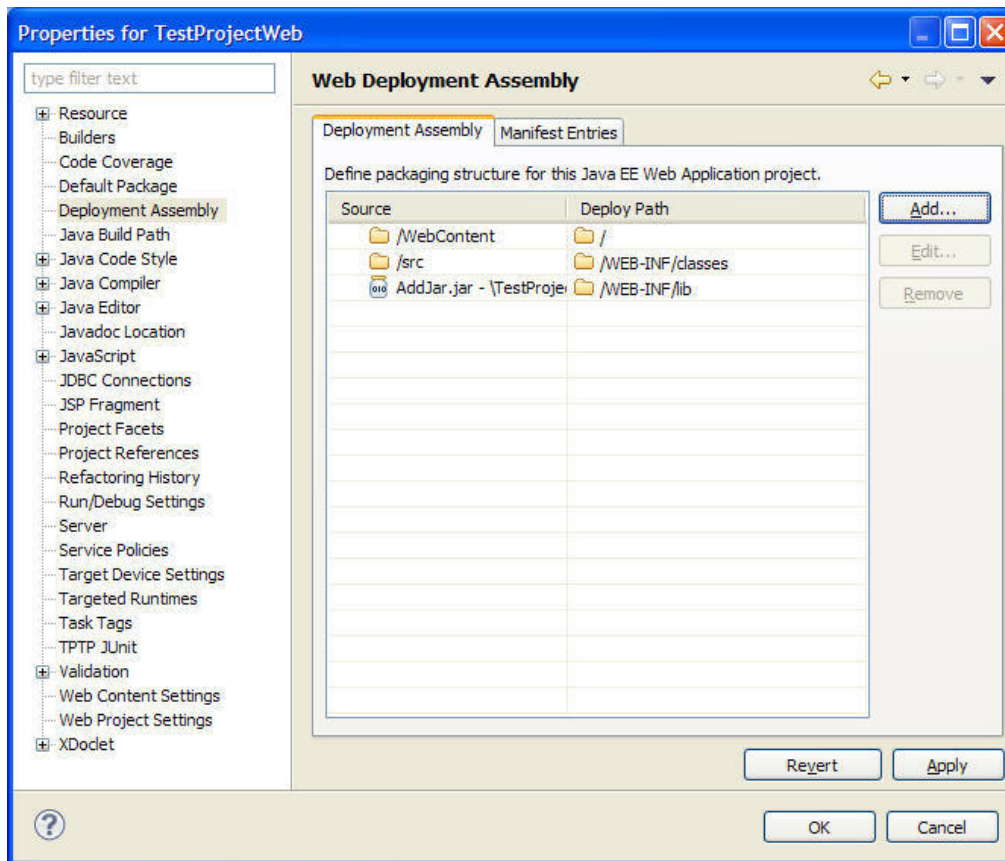
5. If you look at the .classpath file, the JAR files are now either included or excluded from the dependencies:

```

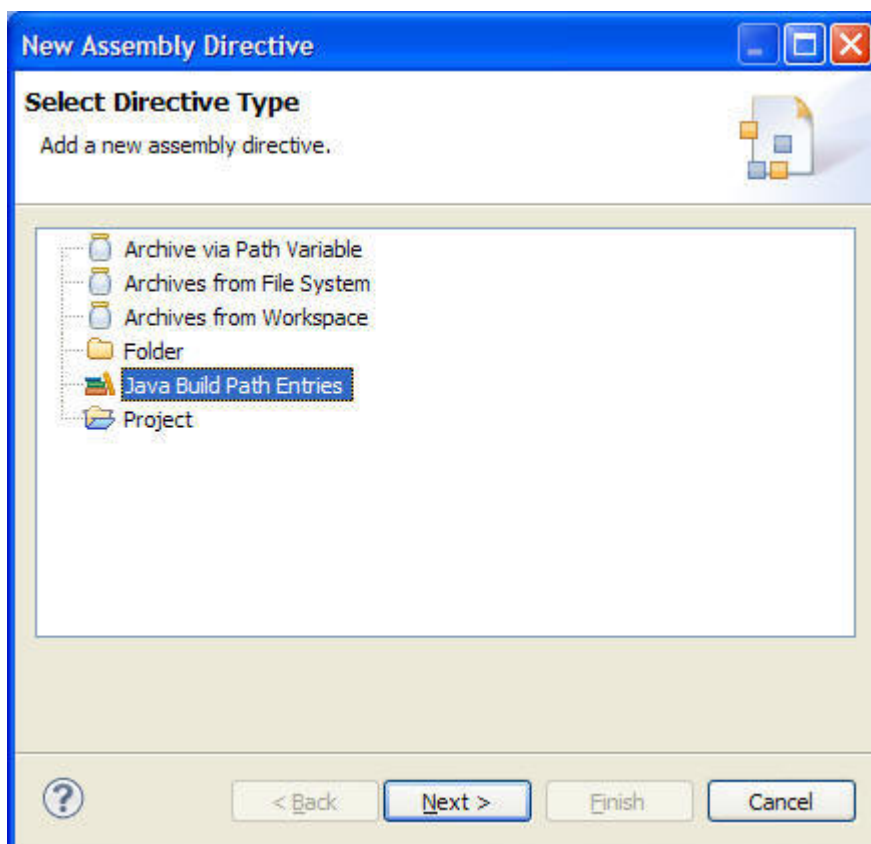
<classpathentry kind="lib" path="/AddJars/AddJars1.jar">
  <attributes>
    <attribute name="org.eclipse.jst.component.dependency" value="/WEB-INF/lib"/>
  </attributes>
</classpathentry>
<classpathentry kind="lib" path="/AddJars/AddJars2.jar">
  <attributes>
    <attribute name="org.eclipse.jst.component.nondependency" value=""/>
  </attributes>

```

6. Right-click the web project, and select **Properties > Deployment Assembly** to see what JAR files are included as a web library dependency:



Also, you can add JAR files as a web library dependency by clicking **Add**:



Parent topic: [Creating web modules](#)

Creating and configuring resource references for Web 2.5, Web 3.0, and Web 3.1

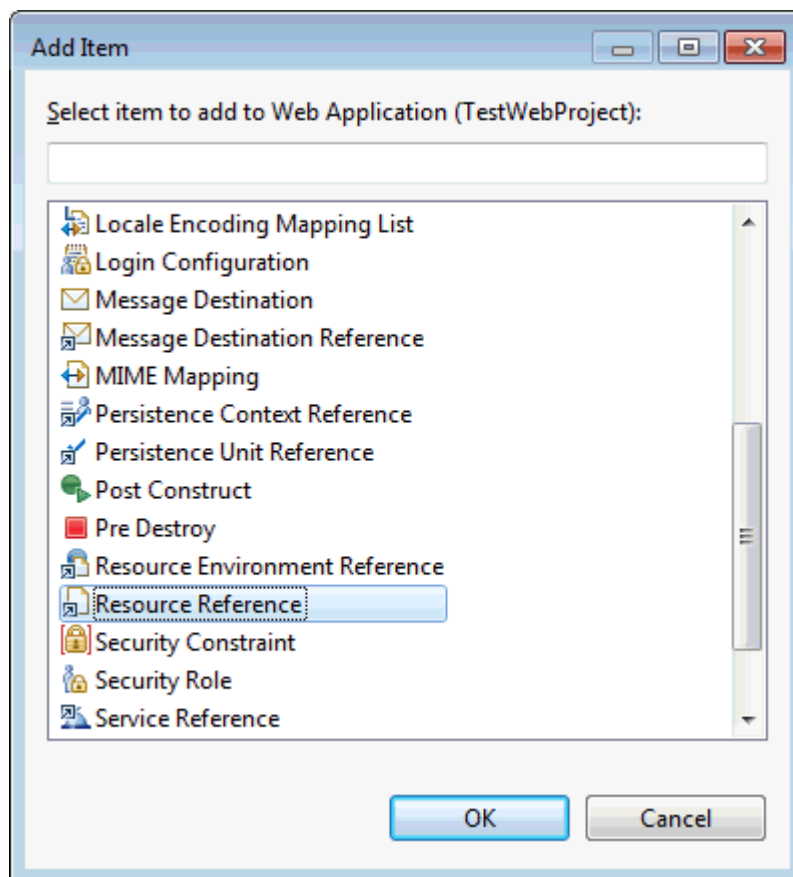
You can create and configure resource references for Web 2.5, Web 3.0, and Web 3.1 projects using the deployment descriptor.

About this task

You need to create a web project using Web 2.5, Web 3.0, and Web 3.1 before you can create and configure resource references. Select **Generate deployment** descriptor in the project creation wizard.

Procedure

1. Expand your web project, and select **WebContent > WEB-INF > web.xml**.
2. Right-click `web.xml` and select **Open with > Web Application Deployment Descriptor Editor**.
3. In the Web application field, select **Add** and select **Resource Reference** and click **OK**:



4. In the **Details** section, provide the details for your resource reference:
 - A. In the **Name** field, provide a name for your resource reference.
 - B. In the **Type** field, provide the type of resource reference.
 - C. In the **Authentication** field, select either **Application** or **Container** for the authentication of your resource reference.
 - D. In the **Sharing Scope** field, select either **Shareable** or **Unshareable** for the sharing scope of your resource reference.
 - E. In the **Description** field, type a description of this resource reference.
5. To view the `web.xml` source code, select **Source**: `<?xml version="1.0" encoding="UTF-8"?>`

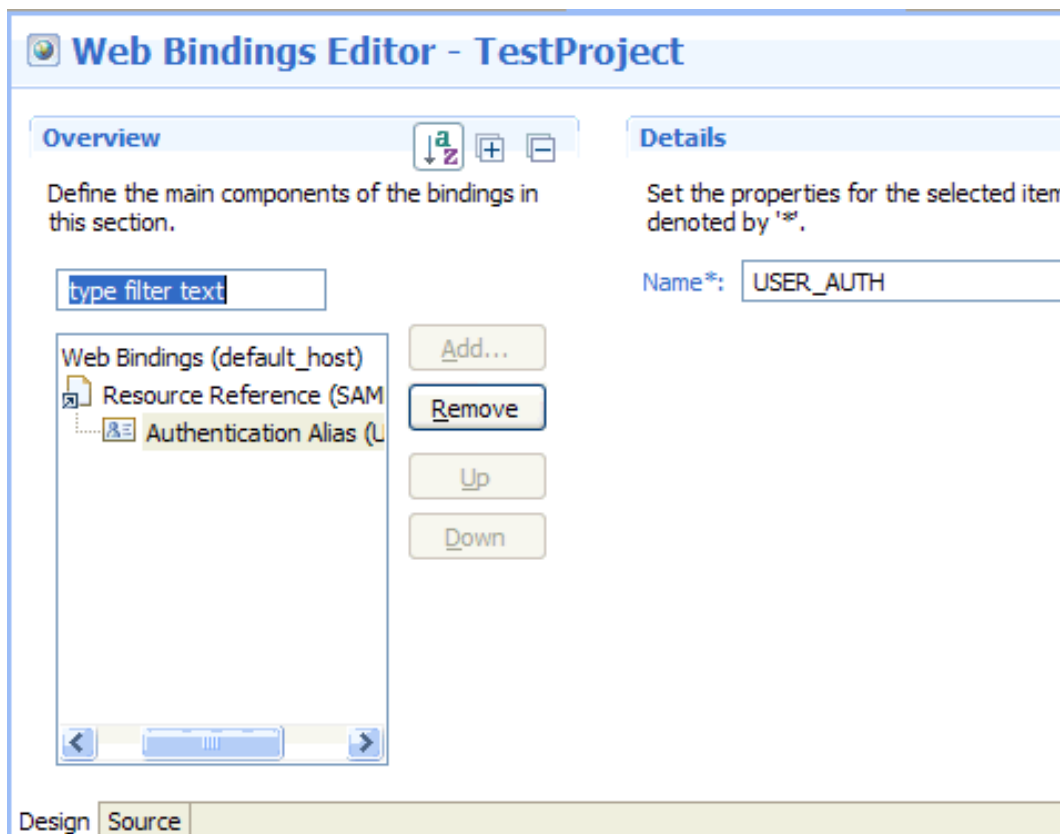
```
<web-app id="WebApp_ID" version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
<servlet>
  <description>
  </description>
  <display-name>
  TestServlet</display-name>
  <servlet-name>TestServlet</servlet-name>
  <servlet-class>test.TestServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>TestServlet</servlet-name>
  <url-pattern>
  /TestServlet</url-pattern>
</servlet-mapping>
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>default.html</welcome-file>
  <welcome-file>default.htm</welcome-file>
  <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
<resource-ref>
  <res-ref-name>SAMPLE</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
</web-app>

```

6. To edit your resource reference, you can make changes in either the Design or the Source view of the deployment descriptor page.
7. To bind this resource reference to a data source on the server, with the JNDI name jdbc/SAMPLE using a JAAS authentication alias called USER_AUTH, edit the `ibm-web-bnd.xml` file and add the following definitions:
 - A. If the `ibm-web-bnd.xml` file does not exist, create the file by right-clicking on the project and then selecting **Java EE tools > Generate WebSphere Bindings Deployment Descriptor**.
 - B. Right-click the `ibm-web-bnd.xml` file and select **Open with > Web Bindings Editor**.
 - C. Add a resource reference:
 1. In the Design view, click **Add**.
 2. In the Add Item window, select **Resource Reference**. Click **OK**.
 3. In the **Name** field, type the name of your resource reference, for example: SAMPLE.
 4. In the **Binding Name** field, type the name of your resource reference, for example: jdbc/SAMPLE.
 - D. In the Design view, highlight your Resource Reference, and click **Add > Authentication Alias**.
 - E. In the Details section, in the **Name** field, provide a name for your Authentication Alias (for example, use the JAAS authentication alias called USER_AUTH).



F. Save the file.

G. To view the `ibm-web-bnd.xml` source code, select **Source**: `<?xml version="1.0" encoding="UTF-8"?>`

```
<web-bnd
  xmlns="http://websphere.ibm.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://websphere.ibm.com/xml/ns/javaee http://websphere.ibm.com/xml/ns/javaee/ibm-web-
bnd_1_0.xsd"
  version="1.0">

  <virtual-host name="default_host" />

  <resource-ref name="SAMPLE" binding-name="jdbc/SAMPLE">
    <authentication-alias name="USER_AUTH" />
  </resource-ref>
</web-bnd>
```

Parent topic: [Creating web modules](#)

Creating web fragment projects

You can use the web fragment project wizard to create web fragment projects in your workspace.

Before you begin

A web fragment is a logical partitioning of the web application in such a way that the frameworks being used within the web application can define all the artifacts without requiring you to edit or add information in the `web.xml`. It can include almost all the same elements that the `web.xml` descriptor uses, with these requirements:

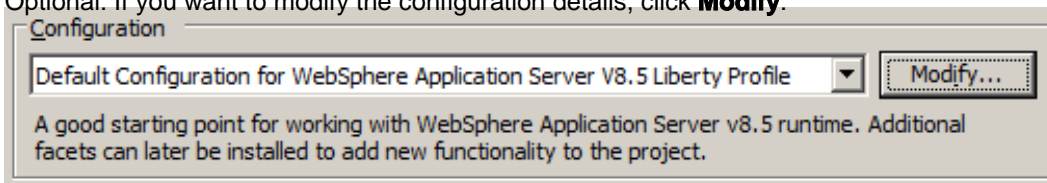
- The top-level element for the descriptor must be `web-fragment`
- The corresponding descriptor file must be called `web-fragment.xml`

If a framework is packaged as a JAR file and has metadata information in the form of deployment descriptor, then the `web-fragment.xml` descriptor must be in the `META-INF/` directory of the JAR file.

A web fragment is a mechanism for either defining or extending the deployment descriptor of a web application by using pluggable library JAR files that contain both the incremental deployment information (in the `web-fragment.xml`) and potentially any related or relevant classes. The web fragment is also packaged as a library (JAR), with the `web-fragment.xml` in the `META-INF` directory. Consequently, the web fragment project is essentially a utility project, with the addition of a web fragment facet to it. The web fragment facet enables you to add relevant context-sensitive functionality to the fragment project.

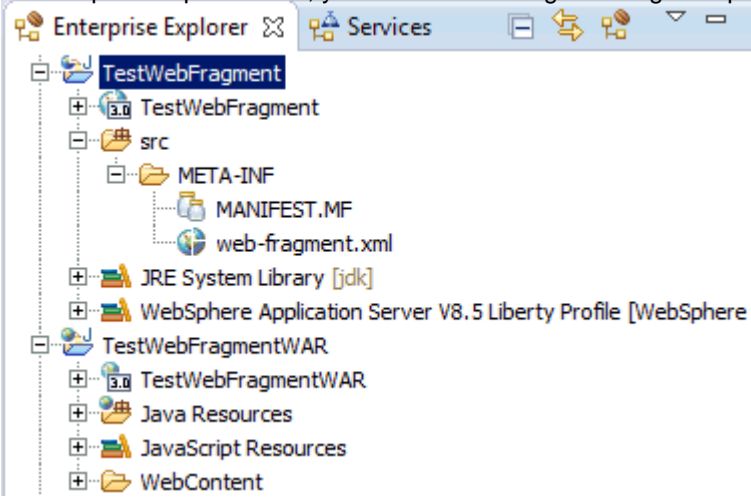
Procedure

1. In the Java™ EE perspective, select **File > New > Project... > Web > Web Fragment Project > Next**. The web fragment wizard opens.
2. In the **Project name** field, type a name for the web fragment project. To change the default **Project location**, clear the **Use default location** checkbox and click the **Browse** button to select a new location. If you specify a non-default project location that is already being used by another project, the project creation fails.
3. The **Target runtime** field is pre-populated with the selection from the enterprise project.
4. Optional: Select a pre-defined project configuration from the **Configuration** drop-down list.
5. Optional: If you want to modify the configuration details, click **Modify**:



- Select one or more project facets from the **Project Facets** list. To specify server runtime environments, click **Runtimes** and select one or more runtimes. After making your selections, you can save your custom configuration by clicking **OK**.
6. Optional: Select the **Add project to Dynamic Web project** check box to add the new module to a web archive (WAR) project. Type a new dynamic web project name or select an existing project from the drop-down list in the **Dynamic Web project name** combination box. Or, click **New** to launch the Dynamic Web Project wizard.
 7. Select **Add project to working sets** to add the web fragment project to an existing working set, or click **Select** to locate a working set. Click **Next**.
 8. On the Configure project for building a Java application page, on the **Source folders on build path** field, click **Add Folder...** to add folders for your source on the build path, or accept the default value (`src`).
 9. In the **Default output folder** field, specify a folder for your output files or accept the default value (`bin`), and click **Finish**.

10. In Enterprise Explorer view, you see the resulting web fragment project folders:



Parent topic: [Creating and configuring Java EE projects using wizards](#)

Creating connector modules

You can use wizards to create a connector module in your Java™ EE project.

Procedure

1. In the Java EE perspective, right-click your enterprise application project and select **New > Connector project**. The Connector project wizard opens.
2. In the **Name** field, type a name for the connector project. To change the default **Project location**, clear the **Use default location** checkbox and click the **Browse** button to select a new location. If you specify a non-default project location that is already being used by another project, the project creation fails.
3. The **Target runtime** field is pre-populated with the selection from the enterprise project.
4. In the **Connector Module version** field, accept the default value.
5. Optional: Select a pre-defined project configuration from the **Configurations** drop-down list.
6. Optional: **Modify Configuration**: If you want to modify the configuration details, click **modify**.
 - Select one or more project facets from the **Project Facets** list. To specify server runtime environments, click **Runtimes** and select one or more runtimes. After making your selections, you can save your custom configuration by clicking **OK**.
7. Optional: Select the **Add project to an EAR module** check box to add the new module to an enterprise module (EAR) project. Type a new project name or select an existing enterprise module project from the drop-down list in the **EAR Project Name** combination box. Or, click **New Project** to launch the New EAR module Project wizard.
8. Click **Next**.
9. On the Connector Project page, in the **Source Folder**: field, specify a folder for your source files by clicking **Add Folder** or accept the default value (connectorModule).
10. In the **Output Folder**: field, specify a folder for your output files or accept the default value (build\classes).
11. Click **Next**. Select **Generate ra.xml deployment descriptor** if you want to create a deployment descriptor.
12. Click **Finish**.

Parent topic: [Creating and configuring Java EE projects using wizards](#)

Deleting Java EE modules

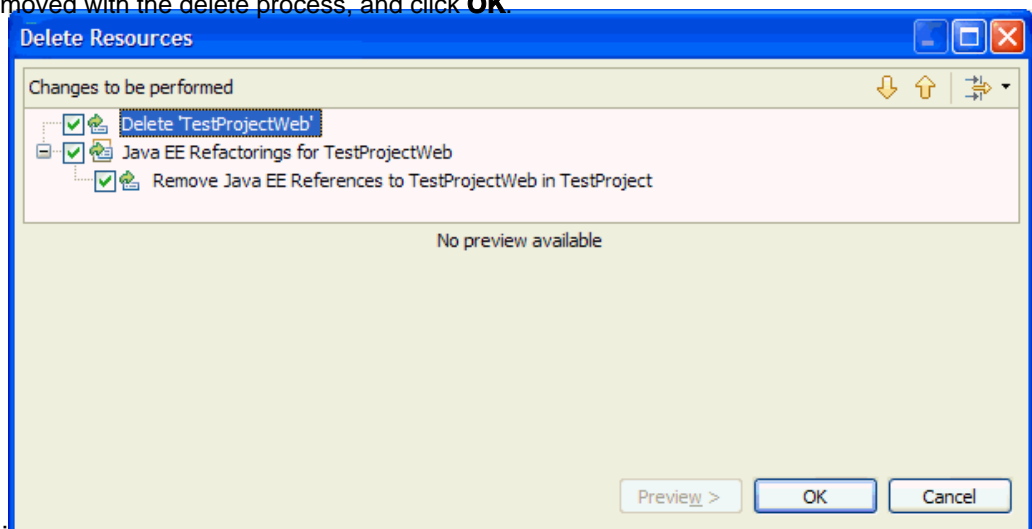
You can delete modules from existing Java™ EE applications.

About this task

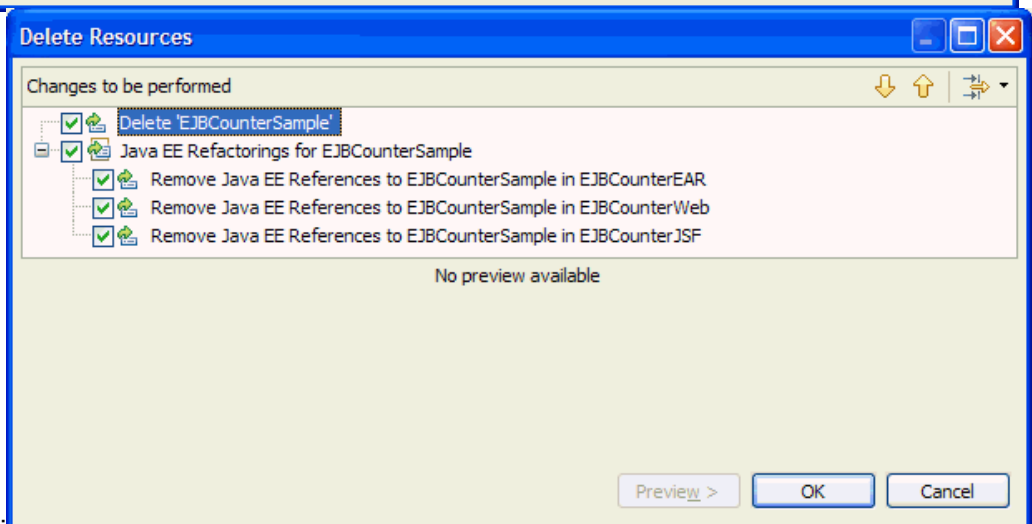
When you create a Java EE project, code is generated in files as part of the process of creating the project. After you delete a module or a bean in a Java EE project, the affected files are also cleaned up. You can delete web modules, EJB modules, Application Client modules, Connector modules, or EJB client, or utility modules.

Procedure

1. Right-click the module that you want to delete, and select **Delete**. Alternatively, you can select the module in the project view, and press `Delete`. A dialog box appears, confirming the deletion.
2. Select **Preview**. A refactoring page appears listing the items that are affected by your delete action. Clear the items that you do not want to be removed with the delete process, and click **OK**.

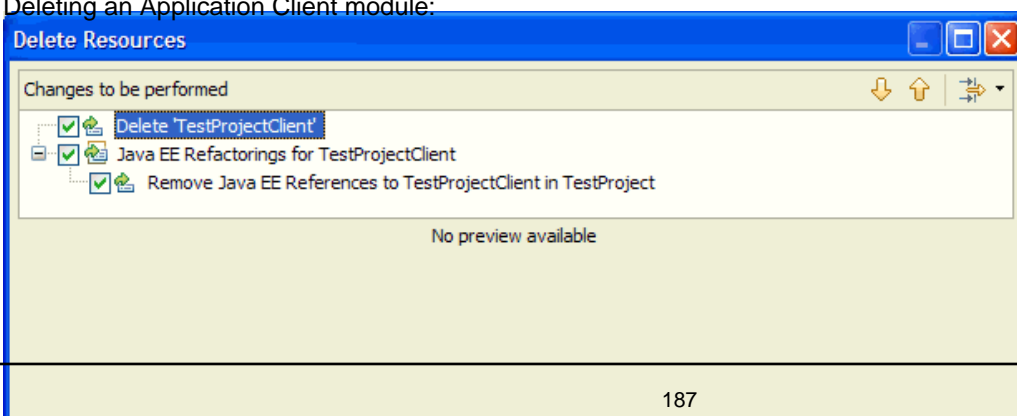


- Deleting a web module:

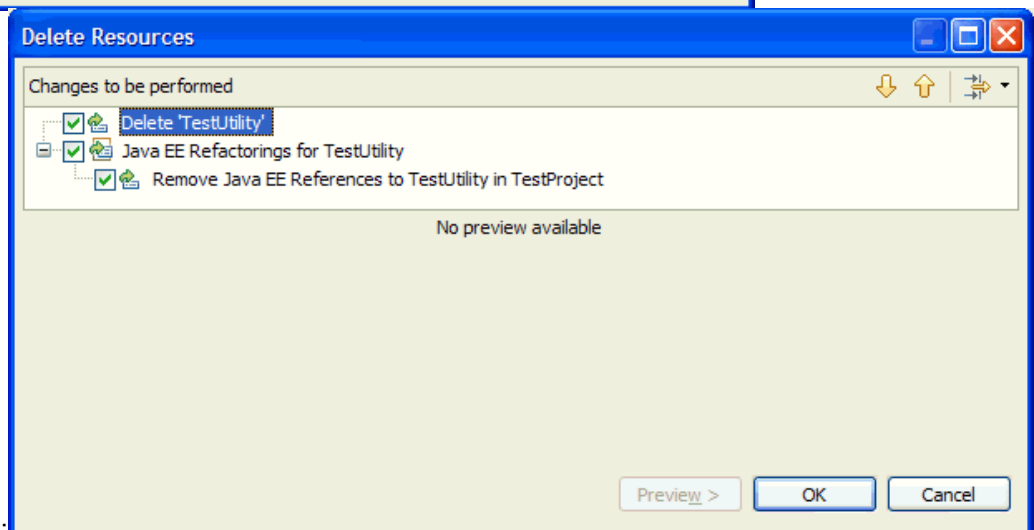
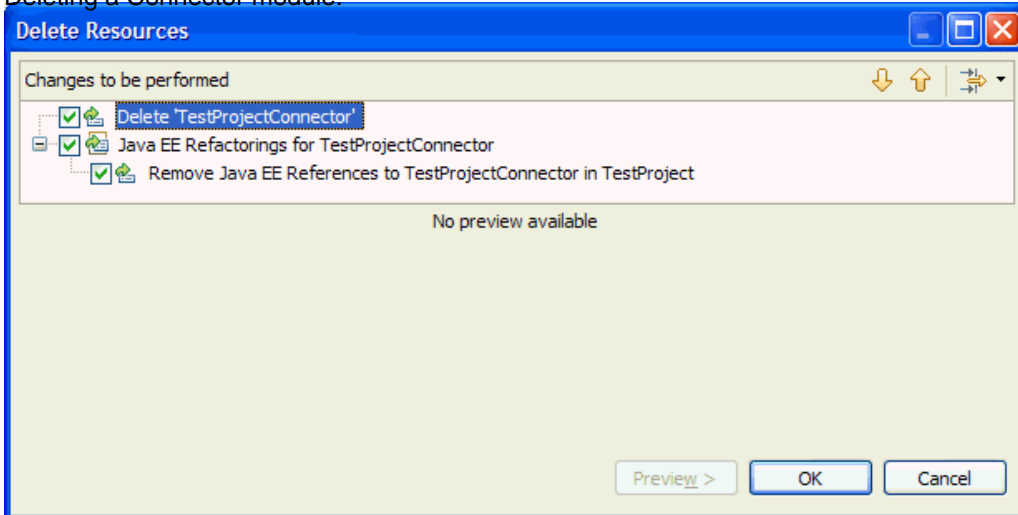


- Deleting an EJB module:

- Deleting an Application Client module:



- Deleting a Connector module:



- Deleting a utility module:

Parent topic: [Creating and configuring Java EE projects using wizards](#)

Importing JAR, WAR, EJB JAR, client application JAR, and RAR files

About this task

You can use the Import wizard to add JAR, WAR, EJB JAR, client application JAR, and RAR files to your enterprise project.

Procedure

1. Right-click your project and select **Import**. Select the type of file you want to import:
 - Application Client JAR file
 - EAR file
 - EJB JAR file
 - Java EE Utility JAR file
 - RAR file
 - Shared EAR file
 - WAR file
2. Follow the steps to import your selected type of file:
 - Application Client JAR file:
 - A. In the Application Client Import page, next to the **Application Client file** field, click **Browse** to locate the file you want to import. When the file is added, the name of the imported project appears in the **Application Client project** field. You can change the name of the project by typing in a different name. Select **Add project to an EAR** and provide a name for the EAR file (the default is derived from your <project_name> + EAR).
 - B. Click **Finish**.
 - EAR file:
 - A. In the Enterprise Application Import page, next to the **EAR file** field, click **Browse** to locate the file you want to import. When the file is added, the name of the imported project appears in the **EAR project** field. You can change the name of the project by typing in a different name. Select your preferred target run time in the **Target Runtime** field, and click **Next**.
 - B. In the Enterprise Application Import page, in the **Utility jars and web libraries** field, and select the additional jars that you want to include in your project. Accept the default for Project locations or modify, and click **Finish**.
 - EJB JAR file:
 - A. In the EJB JAR Import page, next to the **EJB JAR file** field, click **Browse** to locate the file you want to import. When the file is added, the name of the imported project appears in the **EJB project** field. You can change the name of the project by typing in a different name. Select **Add project to an EAR** and provide a name for the EAR file (the default is derived from your <project_name> + EAR).
 - B. Click **Finish**.
 - Java EE Utility JAR:
 - A. On the Utility Jar Import page, in the **Select EAR Project** field, ensure that your project appears, or type a new project name. In the **Select import type** field, select one of the following options:
 - Create Java™ Projects from Utility JARs
 - Create linked Java Projects from Utility JARs
 - Copy Utility JARs into an existing EAR from an external location
 - Create Linked Utility JARs in an existing EAR from an external locationIn the **Project import options** field, you can select **Override Project Root (Specify location below)**, and type an alternative project location, or click **Browse** to locate a directory in your file system. Click **Next**.

B. On the Utility Jar Import page, click **Browse** to locate the director that contains the utility JAR files that you want to import. In the **Utility Jars and Web libraries** field, select the files that you want to import. Click **Finish**.

- RAR file:

A. In the Connector Import page, next to the **Connector file** field, click **Browse** to locate the file you want to import. When the file is added, the name of the imported project appears in the **Connector module** field. You can change the name of the project by typing in a different name. Select **Add project to an EAR** and provide a name for the EAR file (the default is derived from your <project_name> + EAR).

B. Click **Finish**.

- Shared EAR file:

A. In the Shared EAR Import page, next to the **EAR file** field, click **Browse** to locate the file you want to import. When the file is added, the name of the imported project appears in the **EAR project** field. You can change the name of the project by typing in a different name. Select your preferred target run time in the **Target Runtime** field.

B. Click **Finish**.

- WAR file:

A. In the WAR Import page, next to the **WAR file** field, click **Browse** to locate the file you want to import. When the file is added, the name of the imported project appears in the **Web project** field. You can change the name of the project by typing in a different name. Select **Add project to an EAR** and provide a name for the EAR file (the default is derived from your <project_name> + EAR). Click **Next**.

B. On the WAR Import: Web Libraries page, select any **Web libraries** that you want to add to your project. Click **Finish**

Parent topic:[Creating and configuring Java EE projects using wizards](#)

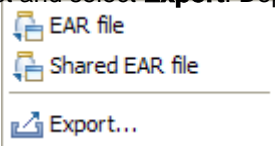

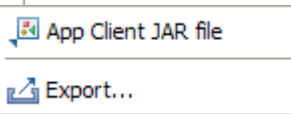
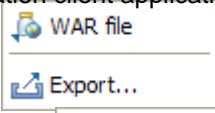
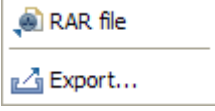
Exporting JAR, WAR, EJB JAR, client application JAR, and RAR files

About this task

You can use the Export wizard to export your enterprise applications as JAR, WAR, EJB JAR, client application JAR, and RAR files.

Procedure

1. Right-click your enterprise project and select **Export**. Depending on the type of your project, select:

- EAR file (enterprise application) 
- EJB JAR file (EJB application) 
- App Client JAR file (application client application) 
- WAR file (Web application) 
- RAR file (connector application) 

2. In the File Export page, your selected project appears. If your project does not appear, type the name of it. In the **Destination** field, type a folder destination, or click **Browse** to locate a destination directory for your exported file. In the **Target Runtime** field, select your preferred target runtime if applicable. Select **Export source files**, if you want to include your source files. Select **Overwrite existing file**, if you want to overwrite an existing file. Click **Finish**.

Parent topic: [Creating and configuring Java EE projects using wizards](#)

Customizing the Liberty class path container

You can customize the contents of the Liberty profile class path container by excluding third-party API libraries, IBM® API libraries, and unrecognized content in your Java™ EE project.

Before you begin

Important: Applicable editions: Liberty profile, Liberty Core

If you do not see the Java EE icon,  , in the workspace, you need to [switch to the Java EE perspective](#).

About this task

Projects that target WebSphere® Application Server Liberty profile automatically have the Liberty profile library added to their class path. This library contains the API classes from the JAR files that are included in the Liberty profile server. The contents of this library depend on the facets that are installed in the project. For example, depending upon which JPA facets are installed, the library includes or excludes certain JAR files to avoid conflicts:

- If the project has the JPA 2.1 facet, the library includes the JAR files for JPA 2.1 and excludes the JAR files for JPA 2.0.
- If the project has the JPA 2.0 facet, the library includes the JAR files for JPA 2.0 and excludes the JAR files for JPA 2.1.
- If the project does not have the JPA facet, the library includes the JAR files for JPA 2.1 and excludes the JAR files from JPA 2.0.

You can alter what is included by default in the class path container by excluding third-party API libraries, IBM API libraries, and unrecognized content.

Procedure

1. Right-click on the project, and select **Properties > Liberty Profile**.
2. Select the API that you want to exclude.
3. Click **OK**.

Parent topic: [Creating and configuring Java EE projects using wizards](#)

Configuring Bean Validation

You can use Bean Validation to use one or more constraints to verify data that is used in application logic. You can use predefined constraints, or you can create your own.

Before you begin

Choose the top-level feature **Web Development Tools** to automatically include the Bean Validation tools.

About this task

Use constraint annotation or XML deployment descriptor files to configure Bean Validation and describe custom constraint declarations.

Procedure

- Configure Bean Validation with constraint annotations.

1. Choose a predefined constraint annotation, or use the meta-annotation `@Constraint` to create a custom constraint and use it in application logic. The interface `ConstraintValidator` must be implemented if you want to use it as a custom constraint class. **Important:** There are 13 predefined constraint annotations, all in the package `javax.validation.constraints`: `@AssertFalse`, `@AssertTrue`, `@DecimalMax`, `@DecimalMin`, `@Digits`, `@Future`, `@Max`, `@Min`, `@NotNull`, `@Null`, `@Past`, `@Pattern`, and `@Size`.
2. Use your chosen annotation before the field declaration to verify data that is used in application logic. For example, method variables can be validated to be not null by using the predefined annotation `@NotNull` before the variable declaration. Validation can be used on fields, method parameters, and at a class level to check more than one data point at a time.

- Configure Bean Validation with XML deployment descriptor files.

1. To create `validation.xml`, click **Java EE Tools > Create Bean Validation configuration file** in WebSphere® Developer Tools.
2. To create new constraint mappings, click **New > Bean Validation > Bean Validation constraints declaration file**. The constraint mappings are declared in the `validation.xml` file by the `<constraint-mapping>` tag. The constraint mapping files don't follow a naming formula. They are automatically placed in the `META-INF/validation` directory of the archive.
3. Use a deployment descriptor editor to configure and declare validation constraints that are used in your application logic.

Parent topic: [Creating and configuring Java EE projects using wizards](#)

Related information:

[Bean Validation specifications](#)

Creating and configuring Java EE modules using annotations

Annotations are a modifier or metadata tag that provide additional data to Java™ classes, interfaces, constructors, methods, fields, parameters, and local variables. With Java EE, you can use annotations to create and configure modules.

- [Java EE with annotations overview](#)

The goal of Java EE 6 platform development is to minimize the number of artifacts that you have to create and maintain, thereby simplifying the development process. Java EE supports the injection of annotations into your source code, so that you can embed resources, dependencies, services, and lifecycle notifications in your source code, without having to maintain these artifacts elsewhere.

- [Scope and placement of annotations](#)

You can add annotations to your source code at the class, method, and field level.

- [Annotations view](#)

The Annotations view detects the annotation types from the metadata of the annotation tag implementation class, shows you any default values, and provides a place to add, edit, and remove annotations.

- [Defining your own annotations](#)

You can use the `@interface` annotation to create your own annotation definition.

- [Adding annotations](#)

You can add annotations into your source code by using the Annotations view or by adding the annotation directly in the Java editor.

- [Editing annotations](#)

You can edit annotations in your source code by directly modifying the annotation in the Java editor or by using the Annotations view.

- [Removing annotations](#)

You can remove annotations in your source code by directly deleting the annotation in the Java editor or by using the Annotations view.

- [Viewing overridden annotations](#)

The Annotations view detects if the value of an annotation or a parameter has been overridden by a deployment descriptor value.

- [Excluding files from annotation scanning](#)

Using the annotation exclusion tool, you can indicate files that you want to exclude from annotation scanning.

- [Types of annotations](#)

Java EE defines a number of types or groups of annotations, which are defined in a number of Java Specification Requests (JSRs).

Parent topic: [Developing Java EE Applications](#)

Related concepts:

[Scope and placement of annotations](#)

[Annotations view](#)

Related tasks:

[Defining your own annotations](#)

[Adding annotations](#)

[Editing annotations](#)

[Removing annotations](#)

[Viewing overridden annotations](#)

[Excluding files from annotation scanning](#)

Java EE with annotations overview

The goal of Java™ EE 6 platform development is to minimize the number of artifacts that you have to create and maintain, thereby simplifying the development process. Java EE supports the injection of annotations into your source code, so that you can embed resources, dependencies, services, and lifecycle notifications in your source code, without having to maintain these artifacts elsewhere.

An annotation is a modifier or metadata tag that provides additional data to Java classes, interfaces, constructors, methods, fields, parameters, and local variables. Annotations replace boilerplate code, common code that is required by certain applications. For example, an annotation can replace the paired interface and implementation required for a web service. Annotations can also replace additional files that programs require, which are maintained separately. For example, annotations can replace the need for a separately maintained deployment descriptor for enterprise Java beans.

Annotations

- Replace descriptors for most purposes
- Remove the need for marker interfaces (like `java.rmi.Remote`)
- Allow application settings to be visible in the component they affect

Java EE provides annotations for the following tasks, among others:

- Developing Enterprise Java bean applications
- Defining and using web services
- Mapping Java technology classes to XML
- Mapping Java technology classes to databases
- Mapping methods to operations
- Specifying external dependencies
- Specifying deployment information, including security attributes

Java EE defines a number of annotations that can be injected into your source code. To declare an annotation, you simply precede the keyword with an "at" sign (`@`). `package com.ibm.counter;`

```
import javax.ejb.Stateless;
```

```
@Stateless
```

```
public class CounterBean {
```

```
}
```

For more information about the categories of annotations that Java EE supports, see [Types of annotations](#).

Parent topic: [Creating and configuring Java EE modules using annotations](#)

Related concepts:

[Learn about Java EE Applications](#)

[Java EE: Overview](#)

[Tools for Java EE development](#)

[Project facets](#)

[Defining Java EE applications](#)

[Enterprise application security](#)

Related tasks:

[Developing Java EE Applications](#)

[Setting Java EE preferences](#)

[Creating and configuring Java EE projects using wizards](#)

[Validating code in enterprise applications](#)

Scope and placement of annotations

You can add annotations to your source code at the class, method, and field level.

Using Annotations

EJB 3.1 and the Java™ persistence API make use of metadata annotations, a feature that was introduced in J2SE 5.0. An annotation consists of the @ sign preceding an annotation type, sometimes followed by a parenthetical list of element-value pairs. The EJB 3.1 specification defines various annotation types, for example:

- `Component-defining annotation`, such as `@Stateless`, which specifies the bean type
- `@Remote` and `@Local` specify whether a bean is remotely or locally accessible
- `@TransactionAttribute` specifies transaction attributes
- `@MethodPermissions`, `@Unchecked`, and `@SecurityRoles` specify security and method permissions

The Java Persistence API adds annotations specific to the creation of entities, for example:

- `@Entity` is a component-defining annotation that specifies that a class is an entity
- `@Table` specifies the data source to be used in the class

Note: JPA mapping annotations (`@Id`, `@Column`, for example) can be applied to both fields and methods, but for any one entity class you can only apply them to one or the other; that is, either all annotations must be applied to fields, or they all must be applied to methods. Fields can only have private, protected or package visibility, and clients of an entity are not allowed to access the fields directly, so you need to define public getters and setters.

- `@MethodPermissions`, `@Unchecked`, and `@SecurityRoles` specify security and method permissions

Scope and placement of annotations

Annotations operate at a class, interface, method, or field level. For example, component-defining annotations (like `@Stateless` or `@Entity`) are class-level annotations and they are inserted in the comments section before the class

declaration: `package com.ibm.websphere.ejb3sample.counter;`

```
import javax.ejb.Stateless;
import javax.interceptor.Interceptors;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
```

```
@Stateless
@Interceptors ( Audit.class )
```

```
public class StatelessCounterBean implements LocalCounter, RemoteCounter {
```

The order of these annotations is not significant; typically, the component-defining annotation is placed before other annotations, but this placement is not required. Method-level and field-level annotations appear within the class or method:

```
public class JPACounterEntity {

    @Id
    private String primaryKey = "PRIMARYKEY";

    private int value = 0;
```

Parent topic: [Creating and configuring Java EE modules using annotations](#)

Related tasks:

[Creating and configuring Java EE modules using annotations](#)

[Defining your own annotations](#)

[Adding annotations](#)

[Editing annotations](#)

[Removing annotations](#)

[Viewing overridden annotations](#)

[Excluding files from annotation scanning](#)

Annotations view

The Annotations view detects the annotation types from the metadata of the annotation tag implementation class, shows you any default values, and provides a place to add, edit, and remove annotations.

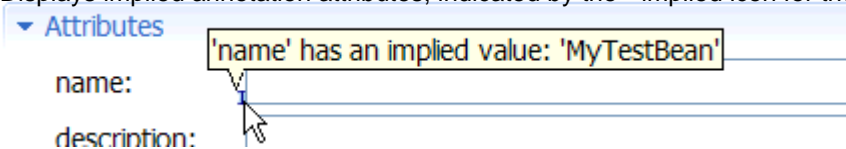
From the Java™ EE perspective, if you do not see the Annotations view, open by selecting **Window > Show View >**

Other > Java > Annotations

Annotations view

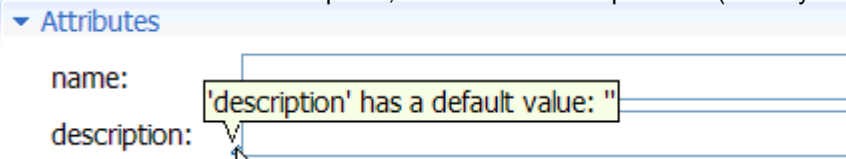
The Annotations view provides a way for you to create, edit, browse, and generally keep track of the annotations that you use in your applications. The Annotations view performs the following functions:

- Displays in an easy-to-navigate tree structure all of your annotations in your Java classes. You can add and remove annotations using the toolbar icons. You can filter the tree by typing a filter value in the **type filter text field**.
- Detects annotation types from the metadata in the annotation tag implementation class to provide rich editing capability.
 - Indicates what attributes can be defined for an annotation, and indicates if an annotation does not define attributes.
 - Provides default validation and user assistance for each annotation.
 - Indicates which attributes are required.
- Displays implied annotation attributes, indicated by the **I** implied icon for the attribute name, in this way:



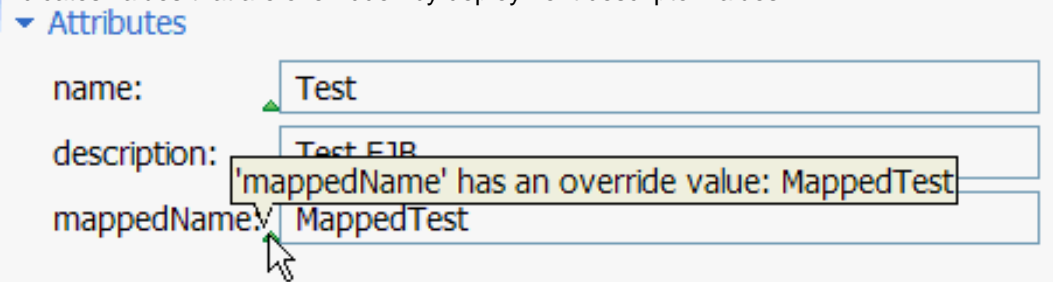
By hovering over the **I**, you can see the implied value for the attribute.

- For attributes that are not required, default values are provided (which you can change):



By hovering over the **Δ**, you can see the default value for the attribute.

- Indicates values that are overridden by deployment descriptor values:



By hovering over the **Δ**, you can see the overridden value for the attribute.

Parent topic: [Creating and configuring Java EE modules using annotations](#)

Related tasks:

[Creating and configuring Java EE modules using annotations](#)

[Defining your own annotations](#)

[Adding annotations](#)

[Editing annotations](#)

[Removing annotations](#)

[Viewing overridden annotations](#)

[Excluding files from annotation scanning](#)

Defining your own annotations

You can use the `@interface` annotation to create your own annotation definition.

Procedure

Use the `@interface` annotation to define your own annotation definition:

- Annotation definitions resemble interface definitions
- Annotation method declarations have neither parameters nor `throws` clauses, and return one of the following elements:
 - primitives
 - String
 - Class
 - enum
 - an array of the previously mentioned types

- Methods can have default values

```
public @interface CreatedBy{
```

```
    String name();
    String date();
    boolean contractor() default false;
}
```

```
@CreatedBy(name = "Mary Smith",date="02/02/2008")
```

```
public class MyClass{...}
```

Results

Meta-annotations: Meta-annotations (annotations of annotations) provide additional information about how an annotation can be used:

- `@Target`
 - Restricts the use of an annotation
 - Single argument must be from Enum `ElementType`
 - {TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL_VARIABLE, ANNOTATION_TYPE}
- `@Retention`
 - Indicates where the annotation information is retained
 - Single argument must be from Enum `RetentionPolicy`
 - {SOURCE, CLASS, RUNTIME}
- `@Documented`
 - Marker for annotations that are to be included in Javadoc
- `@Inherited`
 - Marker for Type annotations that are to be inherited by subtypes

Other built-in annotations:

- `@Overrides`
 - Applied to a method
 - Indicates that the compiler generates an error if the method does not actually override a superclass method.
- `@Deprecated`
 - Applied to a method
 - Indicates that the compiler generates a warning when the method is used externally
- `@SuppressWarnings`
 - Applies to a type or a method

- Indicates that the compiler suppresses warnings for that element and all subelements `@Deprecated`

```
public void oldMethod() {...}
```

```
@SuppressWarnings
```

```
public void yesIknowIuseDeprecatedMethods() {...}
```

Parent topic: [Creating and configuring Java EE modules using annotations](#)

Related concepts:

[Scope and placement of annotations](#)

[Annotations view](#)

Related tasks:

[Creating and configuring Java EE modules using annotations](#)

[Adding annotations](#)

[Editing annotations](#)

[Removing annotations](#)

[Viewing overridden annotations](#)

[Excluding files from annotation scanning](#)

Adding annotations

You can add annotations into your source code by using the Annotations view or by adding the annotation directly in the Java™ editor.

About this task

Java EE 5 and later support the injection of annotations into your source code so that you can embed resources, dependencies, services, and lifecycle notifications in your source code, without having to maintain these artifacts elsewhere. Annotations simplify the development and configuration of enterprise applications.

Procedure

1. Determine which annotations to add to your source code. See documentation about [Java EE](#), [security](#), [web application](#), or [web service](#) annotations.
2. Add the annotations by using the Annotations view or by manually editing the source code. See topics about adding annotations for instructions.

- **Adding annotations using the Annotations view**

You can add annotations in your source code by using the Annotations view.

- **Adding annotations manually**

You can add annotations in your source code by directly adding the annotation in the Java editor.

Parent topic: [Creating and configuring Java EE modules using annotations](#)

Related concepts:

[Scope and placement of annotations](#)

[Annotations view](#)

Related tasks:

[Creating and configuring Java EE modules using annotations](#)

[Defining your own annotations](#)

[Editing annotations](#)

[Removing annotations](#)

[Viewing overridden annotations](#)

[Excluding files from annotation scanning](#)

Adding annotations using the Annotations view

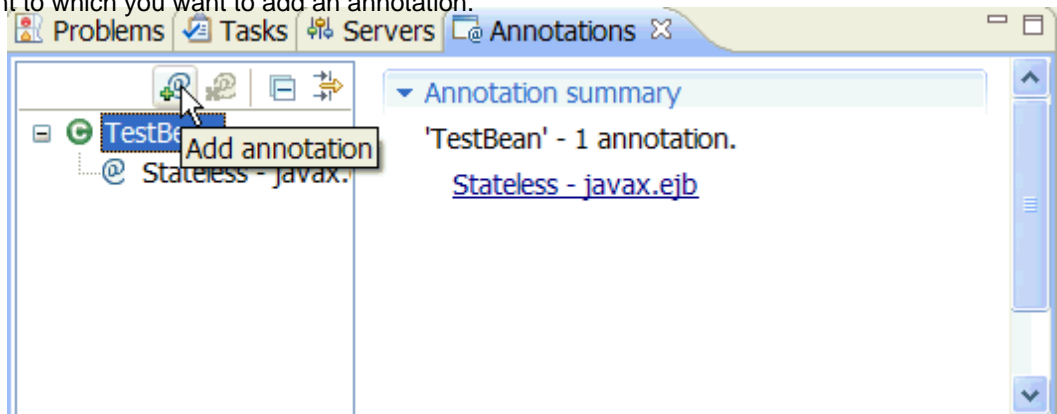
You can add annotations in your source code by using the Annotations view.

Before you begin

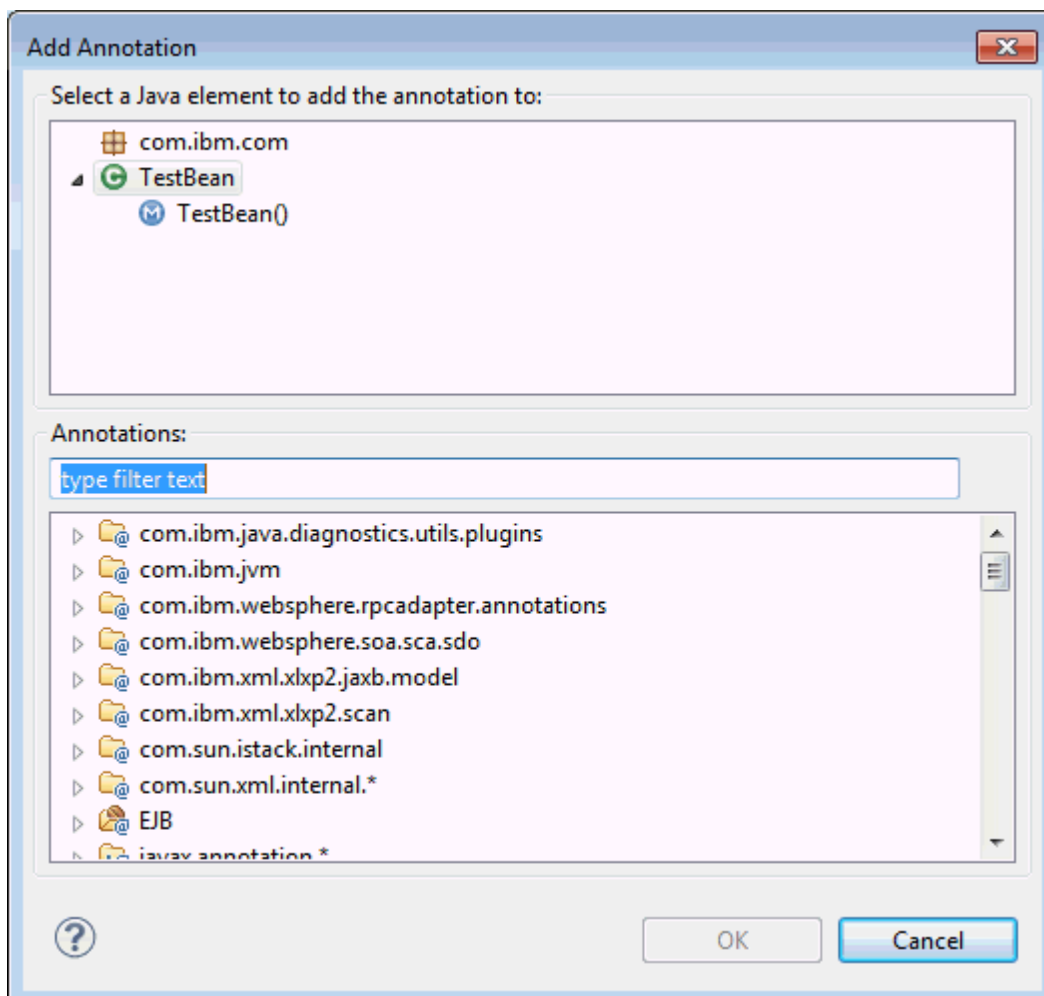
In the Java™ EE perspective, if the Annotations view does not appear in the workspace, you need to add the Annotations view. For more information, see [Annotations view](#).

Procedure

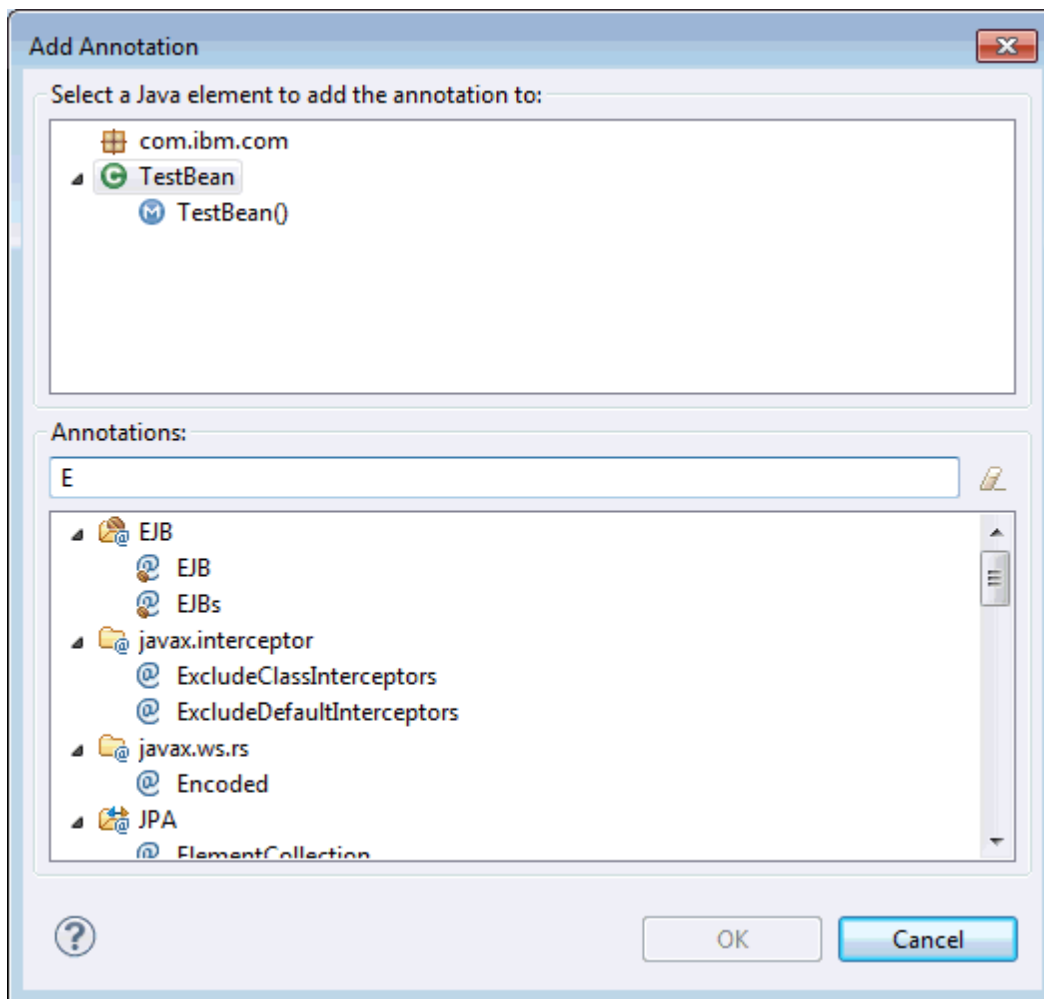
1. In the Enterprise Explorer view, double-click your Java class to open the file in the Java editor.
2. Click the **Annotations view** tab.
3. Select the Java element to which you want to add an annotation.




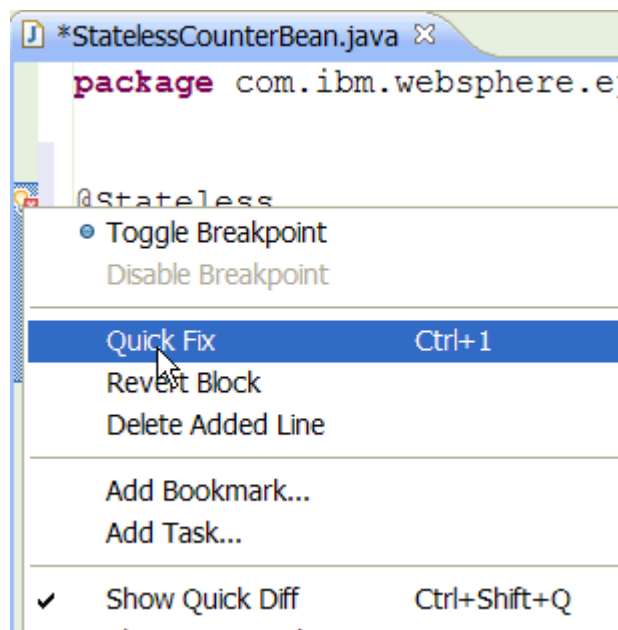
4. Click **Add annotation**.
5. In the Add annotation page, select the annotation that you want to add:



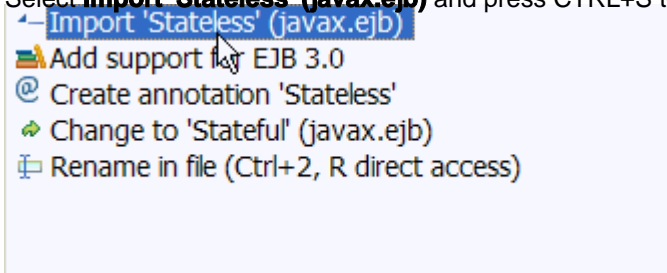
6. To filter the annotations, type a letter or term into the **type filter text** field. You can use "*", "?", or camel case:



- 7. Select the annotation that you want to insert into your Java class, and click **OK**
- 8. You see in the Java class the @Stateless annotation is added to your code.
- 9. When you press CTRL+S to save, you can see a quick fix icon  next to the @Stateless line
- 10. Right-click the quick fix icon and select **Quick Fix**:



11. Select **Import 'Stateless' (javax.ejb)** and press CTRL+S to save:



12. You see in the Java class that your annotation is added to the code. Highlight the annotation, and the details for this annotation appear in the Annotation view.

13. The attributes pane shows both the default values (which you can change) and assigned values.

Parent topic: [Adding annotations](#)

Adding annotations manually


You can add annotations in your source code by directly adding the annotation in the Java™ editor.

Procedure

1. In the Enterprise Explorer view, double-click your Java class to open the file in the Java editor.
2. Place your cursor in the section of your class (class level, method level) where you want to add an annotation.
3. Type the symbol @ followed by the name of your annotation: @Session

```
@Interceptors
```

Note: The order in which you place the annotations is not significant, though generally, the component-defining annotation appears before other types of annotations. What is important is that annotations are placed in the appropriate place in the source file; class-level annotations must appear before the class declaration; method-level annotations are introduced before the method declaration, and field-level annotations are applied to the field itself.

4. If the annotation requires parameters, an error appears in the margin. You can right-click the error and select Quick Fix () to import required jars or to add required parameters.

Parent topic: [Adding annotations](#)

Editing annotations

You can edit annotations in your source code by directly modifying the annotation in the Java™ editor or by using the Annotations view.

Before you begin

In the Java EE perspective, if the Annotations view does not appear in the workspace, you need to add the Annotations view. For more information, see [Annotations view](#).

- **Editing annotations using the Annotations view**

You can edit annotations in your source code by directly modifying the annotation in the Java editor or by using the Annotations view.

- **Editing annotations manually**

You can edit annotations in your source code by directly modifying the annotation in the Java editor or by using the Annotations view.

Parent topic: [Creating and configuring Java EE modules using annotations](#)

Related concepts:

[Scope and placement of annotations](#)

[Annotations view](#)

Related tasks:

[Creating and configuring Java EE modules using annotations](#)

[Defining your own annotations](#)

[Adding annotations](#)

[Removing annotations](#)

[Viewing overridden annotations](#)

[Excluding files from annotation scanning](#)

Editing annotations using the Annotations view

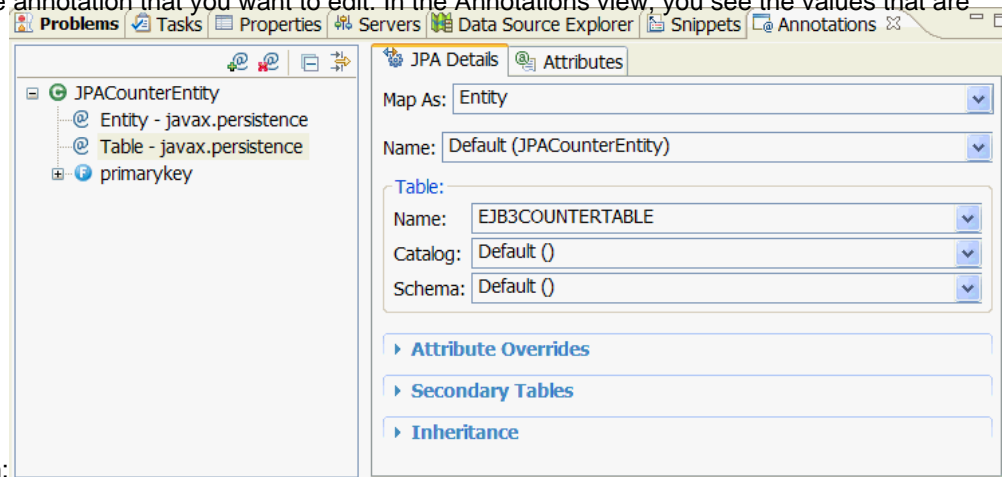
You can edit annotations in your source code by directly modifying the annotation in the Java™ editor or by using the Annotations view.

Before you begin

In the Java EE perspective, if the Annotations view does not appear in the workspace, you need to add the Annotations view. For more information, see [Annotations view](#).

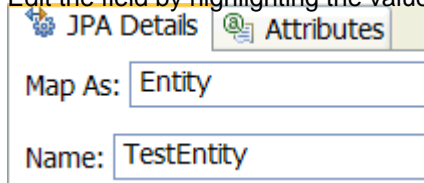
Procedure

1. In the Enterprise Explorer view, double-click your Java class to open the file in the Java editor.
2. Click the **Annotations view** tab.
3. In the Java editor, highlight the annotation that you want to edit. In the Annotations view, you see the values that are



associated with the annotation:

4. Edit the field by highlighting the value in the text box and typing the value that you want:



In this example, the default value of the **Name** field was changed from `Default (JPACounterEntity)` to `TestEntity`.

5. When you press CTRL+S to save, the change is reflected in the Java class:

```
package com.ibm.websphere.ejb3sample.counter;

import javax.persistence.Entity;

@Entity(name="TestEntity")
@Table(name="EJB3COUNTERTABLE")
```

Parent topic: [Editing annotations](#)

Editing annotations manually

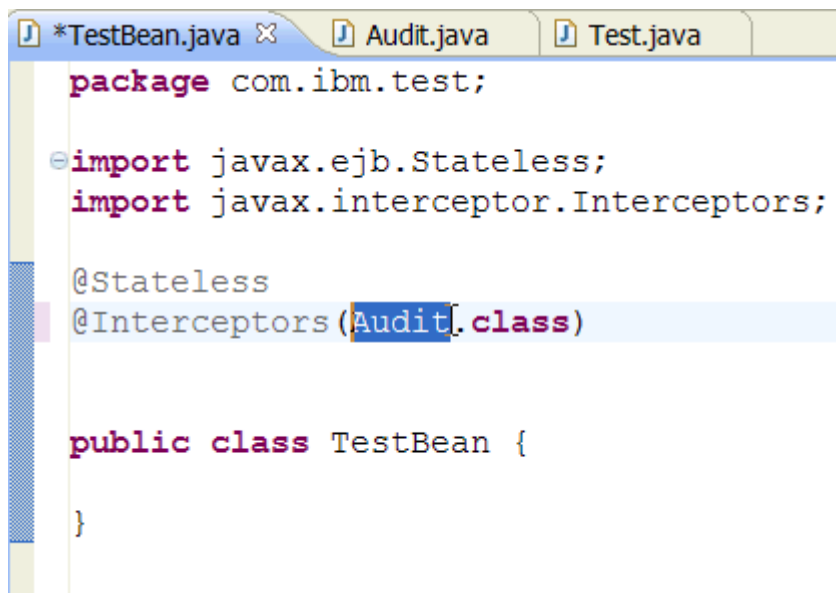
You can edit annotations in your source code by directly modifying the annotation in the Java™ editor or by using the Annotations view.

Before you begin

In the Java EE perspective, if the Annotations view does not appear in the workspace, you need to add the Annotations view. For more information, see [Annotations view](#).

Procedure

1. In the Enterprise Explorer view, double-click your Java class to open the file in the Java editor.
2. In the Java editor, highlight the annotation that you want to edit. You can change the annotation by typing in a different value. For example, you can change `@Stateless` to `@Stateful` to change an EJB from a Stateless Java bean to a Stateful one.
3. You can also edit the values of the parameters that are associated with the annotation, by highlighting the value and typing the value that you want:



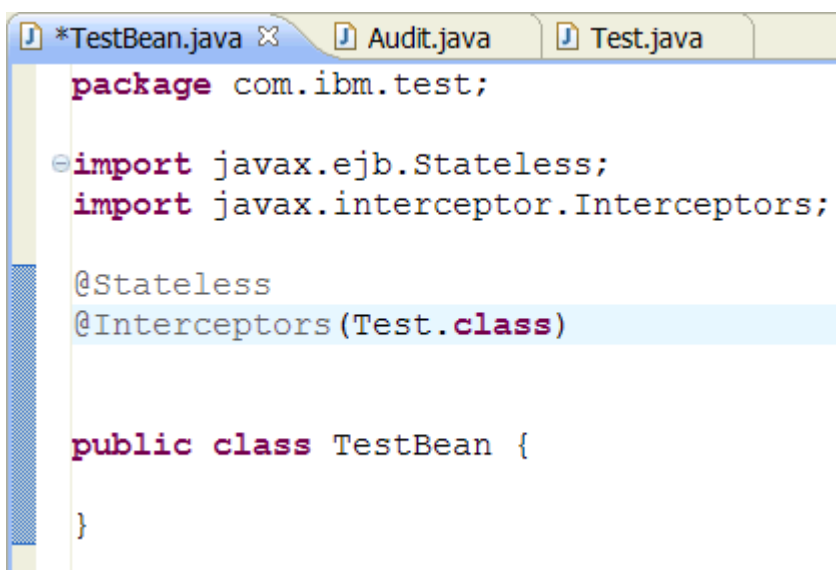
```
package com.ibm.test;

import javax.ejb.Stateless;
import javax.interceptor.Interceptors;

@Stateless
@Interceptors(Audit.class)

public class TestBean {

}
```



```
package com.ibm.test;

import javax.ejb.Stateless;
import javax.interceptor.Interceptors;

@Stateless
@Interceptors(Test.class)

public class TestBean {

}
```

In this example, the default value of the attribute was changed from `Audit.class` to `Test.class`.

Parent topic: [Editing annotations](#)

Removing annotations

You can remove annotations in your source code by directly deleting the annotation in the Java™ editor or by using the Annotations view.

- [Removing annotations using the Annotations view](#)

You can remove annotations in your source code by using the Annotations view.

- [Removing annotations manually](#)

You can remove annotations in your source code by directly deleting the annotation in the Java editor or by using the Annotations view.

Parent topic: [Creating and configuring Java EE modules using annotations](#)

Related concepts:

[Scope and placement of annotations](#)

[Annotations view](#)

Related tasks:

[Creating and configuring Java EE modules using annotations](#)

[Defining your own annotations](#)

[Adding annotations](#)

[Editing annotations](#)

[Viewing overridden annotations](#)

[Excluding files from annotation scanning](#)

Removing annotations using the Annotations view

You can remove annotations in your source code by using the Annotations view.

Before you begin

In the Java™ EE perspective, if the Annotations view does not display in the workspace, add the Annotations view. For more information, see [Annotations view](#).

Procedure

1. In the Enterprise Explorer view, double-click your Java class to open the file in the Java editor.
2. Click the **Annotations view** tab.
3. In the navigation tree, navigate to the annotation that you want to delete.
4. Click the delete annotation icon from the navigation tree.
5. Save your file by pressing CTRL+S.

Parent topic: [Removing annotations](#)

Removing annotations manually

You can remove annotations in your source code by directly deleting the annotation in the Java™ editor or by using the Annotations view.

Procedure

1. In the Enterprise Explorer view, double-click your Java class to open the file in the Java editor.
2. Highlight the annotation that you want to remove, and right-click and select **Cut**. You might need to remove unused imports following the delete operation. If a warning icon appears next to an import statement, right-click the icon, and select **Quick Fix**. Select **Remove unused imports**.
3. Save your file by pressing CTRL+S.

Parent topic: [Removing annotations](#)

Viewing overridden annotations

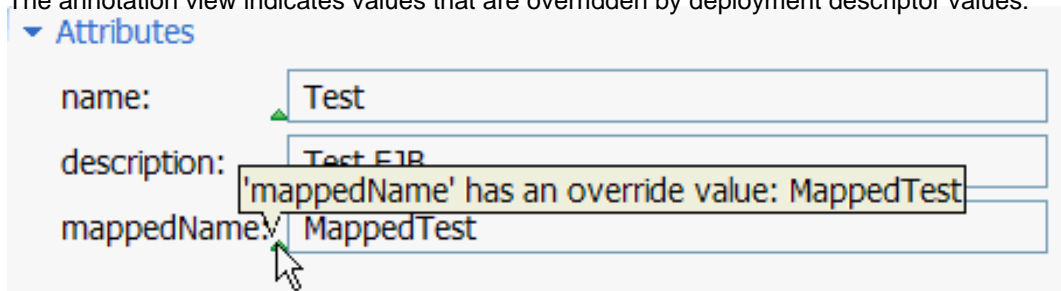
The Annotations view detects if the value of an annotation or a parameter has been overridden by a deployment descriptor value.

Before you begin

In the Java™ EE perspective, if the Annotations view does not display in the workspace, add the Annotations view. For more information, see [Annotations view](#).

Procedure

1. In the Enterprise Explorer view, double-click your Java class to open the file in the Java editor.
2. Click the **Annotations view** tab.
3. Highlight the annotation that you want to view.
4. The annotation view indicates values that are overridden by deployment descriptor values:



By hovering over the **Override** icon (▲), you can see the default value for the attribute.

Parent topic: [Creating and configuring Java EE modules using annotations](#)

Related concepts:

[Scope and placement of annotations](#)

[Annotations view](#)

Related tasks:

[Creating and configuring Java EE modules using annotations](#)

[Defining your own annotations](#)

[Adding annotations](#)

[Editing annotations](#)

[Removing annotations](#)

[Excluding files from annotation scanning](#)

Excluding files from annotation scanning

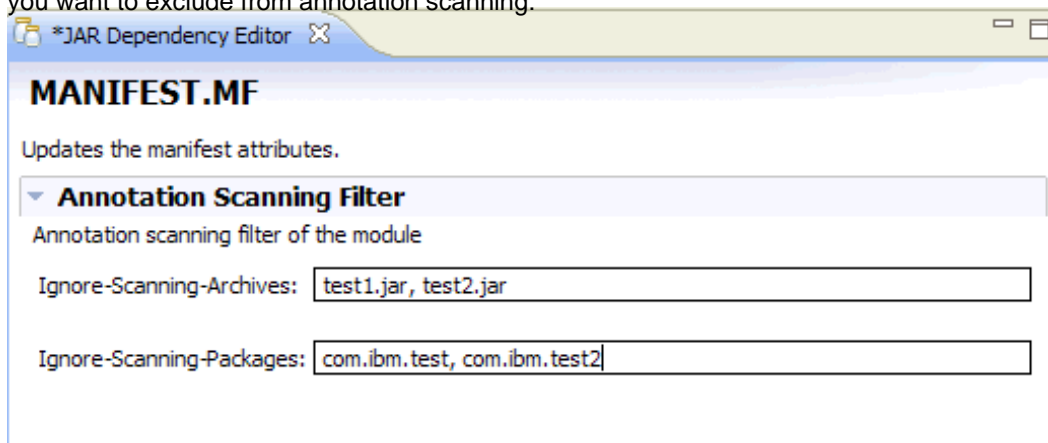
Using the annotation exclusion tool, you can indicate files that you want to exclude from annotation scanning.

Procedure

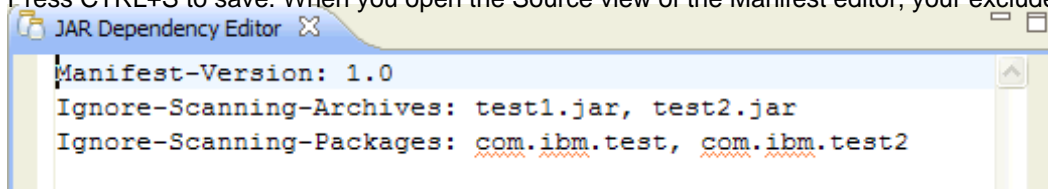
1. To use the annotation exclusion tool:

- A. **In an EAR project:** In the Enterprise Explorer view, right-click your EAR file, and select Java™ EE Tools > Open Manifest.
- B. **In an EJB or web project:** In the Enterprise Explorer view, right-click your MANIFEST.MF file (in the META-INF folder under the ejbModule folder in EJB projects and under the WebContent folder of web projects), and select **Open With > Manifest Editor**.

2. In the Design view of the Manifest editor, in the **Annotation Scanning Filter** section, type the jars and packages that you want to exclude from annotation scanning:



3. Press CTRL+S to save. When you open the Source view of the Manifest editor, your excluded files appear:



Parent topic: [Creating and configuring Java EE modules using annotations](#)

Related concepts:

[Scope and placement of annotations](#)

[Annotations view](#)

Related tasks:

[Creating and configuring Java EE modules using annotations](#)

[Defining your own annotations](#)

[Adding annotations](#)

[Editing annotations](#)

[Removing annotations](#)

[Viewing overridden annotations](#)

Types of annotations

Java™ EE defines a number of types or groups of annotations, which are defined in a number of Java Specification Requests (JSRs).

For information about Java EE 6, see the official specification: [JSR 316: Java™ Platform, Enterprise Edition 6 \(Java EE 6\) Specification](#).

For information about Java EE 5, see the official specification: [JSR 244: Java Platform, Enterprise Edition 5 \(Java EE 5\) Specification](#).

Parent topic: [Creating and configuring Java EE modules using annotations](#)

Defining Java EE applications

You can use the tools in the workspace to define the module dependencies in the Java™ EE project in your workspace.

- [Using the application deployment descriptor editor](#)

You can create deployment descriptors and manage the WebSphere® extensions and bindings in your enterprise applications. You can also change your module dependencies using the deployment descriptor editor.

- [Specifying dependent JAR files or modules](#)

You can use the Java EE Module Dependencies page to specify JAR files or modules that are required by a module. The dependencies are defined in the MANIFEST.MF file for your module.

- [Exporting and importing binary modules](#)

You can add binary modules into and remove binary modules from your workspace.

- [Java EE deployment assembly](#)

Deployment assembly property pages allow you to add flexible resource and dependency mapping to your applications.

Parent topic: [Developing Java EE Applications](#)

Related tasks:

[Generating deployment descriptors](#)

[Generating WebSphere extensions and bindings deployment descriptors](#)

[Specifying dependent JAR files or modules](#)

[Exporting and importing binary modules](#)

Using the application deployment descriptor editor

You can create deployment descriptors and manage the WebSphere® extensions and bindings in your enterprise applications. You can also change your module dependencies using the deployment descriptor editor.

- **Application deployment descriptor editor**

While deployment descriptors are not required since Java EE 5 as they were in J2EE 1.4 and earlier specifications, you can include deployment descriptors in your enterprise applications, and change to your module dependencies using the deployment descriptor editor.

- **Generating deployment descriptors**

You can create deployment descriptors to manage the dependencies for your enterprise projects.

- **Generating WebSphere extensions and bindings deployment descriptors**

You can create deployment descriptors to manage the WebSphere® extensions and bindings for your enterprise application and web projects.

Parent topic: [Defining Java EE applications](#)

Application deployment descriptor editor

While deployment descriptors are not required since Java™ EE 5 as they were in J2EE 1.4 and earlier specifications, you can include deployment descriptors in your enterprise applications, and change to your module dependencies using the deployment descriptor editor.

The application deployment descriptor editor includes scrollable pages and collapsible sections that represent the various properties and settings in the deployment descriptor (`application.xml`) and other metadata that is written to bindings and extensions files. The editor is dynamic, and sections and pages are created based on the application deployment descriptor version and the workbench capabilities that are enabled. To open the editor, right-click the deployment descriptor for your project and select **Open With > Deployment Descriptor Editor**.

Deployment descriptors are not automatically generated when you create your enterprise project, unless you select **Generate Deployment Descriptor** in the project creation wizard. For more information on creating deployment descriptors, see [Generating deployment descriptors](#)

The deployment descriptor editor contains two pages, the Design page and the Source page. The Design page contains a number of sections, including Overview, General Information, and Icons. Collapsing a section hides the content, but leaves the heading information. This is useful in filtering through the data and properties. The editor remembers the sections that you collapse when you close and reopen the editor. The Source page contains the source for your application, the `application.xml`.

Design page

The Design page in the application editor provides a quick summary of the contents in the application deployment descriptor. It includes the following sections: General Information, Modules, Security Roles, Icons, and WebSphere® Extensions.

- Overview section

- The Overview section displays the names of the modules that are defined for the application, and provides a quick link to the Module page of the editor. You can use this section to add, edit, browse, and remove EJB, Web, and Application Client modules from the enterprise application. When you select a module in the list, its attributes are displayed on the fields of the pane. The list of fields changes dynamically to match the type of module selected.

- General Information section

- Use the General Information section to view the display name and description for the enterprise application, as stored in the `application.xml` file.

- Icons section

- Use the Icons section to choose icons that represent your enterprise application. These icons are used for identification on the server. In order to use an icon, you must first import the graphic file into the enterprise application project (basically, it must be contained inside the EAR file in order for it to be found at deploy time). Once the file is imported into the project, you are able to select it within the icon dialog on the application deployment descriptor editor. If you do not import the file into the project, you do not see any icons within the dialogs.

- Actions section

- The Actions section provides links for you to perform the following actions:
 - **Manage Utility Jars:** Use this link to add a Java project as a utility JAR file that can be used by modules in the enterprise application. For each Java project, a utility JAR is created when the EAR file is exported.

- WebSphere Deployment Descriptors

- The Actions section provides links for you to perform the following actions:
 - **Open WebSphere Programming Model Extensions Descriptor :** (For applications that target WebSphere Application Server) The WebSphere bindings section provides a place to add WebSphere Programming Model Extensions.
 - **Open WebSphere Bindings:** (For applications that target WebSphere Application Server) The WebSphere bindings section provides a place to add users and groups to the security roles.

- **Open WebSphere Extensions:** (For applications that target WebSphere Application Server) The WebSphere Extensions section provides a place to add WebSphere Extensions.

If you do not have WebSphere bindings or extensions files created for your project, when you click the action link, a message appears stating that you do not have these files. For information about how to create WebSphere Extensions and Bindings deployment descriptors, see [Generating WebSphere extensions and bindings deployment descriptors](#)

Source page

Use the Source page to view and modify the `application.xml` file directly. The XML on the source page changes dynamically when the deployment descriptor is edited, and the other pages of the application deployment descriptor editor reflect changes that you make on the Source page. Editing the XML source is not the best way to edit the deployment descriptor; use the Design page of the editor to make your changes.

Parent topic: [Using the application deployment descriptor editor](#)

Related tasks:

[Generating deployment descriptors](#)

[Generating WebSphere extensions and bindings deployment descriptors](#)

[Specifying dependent JAR files or modules](#)

[Exporting and importing binary modules](#)

Generating deployment descriptors

You can create deployment descriptors to manage the dependencies for your enterprise projects.

About this task

You can create deployment descriptors for your enterprise applications, either by selecting **Generate Deployment Descriptor** in the project or module creation wizard, or you can add it later:

Procedure

1. Right-click your enterprise application project.
2. Select **Java EE Tools > Generate Deployment Descriptor Stub**
3. The deployment descriptor file for your project or module is created in your project.

Parent topic: [Using the application deployment descriptor editor](#)

Related concepts:

[Application deployment descriptor editor](#)

Related tasks:

[Defining Java EE applications](#)

Generating WebSphere extensions and bindings deployment descriptors

You can create deployment descriptors to manage the WebSphere® extensions and bindings for your enterprise application and web projects.

Procedure

1. Right-click your enterprise application or web project. Select **Java EE Tools**, and then select one of the following options:
 - Generate WebSphere Bindings Deployment Descriptor
 - Generate WebSphere Extensions Deployment Descriptor
 - Generate WebSphere Programming Model Extensions Deployment Descriptor
2. Alternatively, open the deployment descriptor for your project, and in the WebSphere Deployment Descriptor section, select one of the links. If the descriptor that you select does not exist, a message appears asking if you want to create

▼ **WebSphere Deployment Descriptors**

- 🔗 Open [WebSphere Bindings Descriptor](#)
- 🔗 Open [WebSphere Extensions Descriptor](#)
- 🔗 Open [WebSphere Programming Model Extensions](#)

one. Click **Yes**.

3. The deployment descriptor file for your project or module is created in your project.

Parent topic: [Using the application deployment descriptor editor](#)

Related concepts:

[Application deployment descriptor editor](#)

Related tasks:

[Defining Java EE applications](#)

Specifying dependent JAR files or modules

You can use the Java™ EE Module Dependencies page to specify JAR files or modules that are required by a module.

The dependencies are defined in the MANIFEST.MF file for your module.

About this task

When you are specifying required JAR files or modules, you first specify the enterprise application (EAR) that your project is a part of. Typically, the project is referenced by one EAR project in the workspace. However, it is possible that you have multiple enterprise applications that contain a reference to the same module or utility JAR project. If so, then ensure that you give the JAR or module the same Uniform Resource Identifier (URI) in each application, so that the class path is valid for all applications.

It is also possible that your module is a standalone project and is not currently referenced by any enterprise application. In this case, since there is no enterprise application scope that is defined, you cannot use the Manifest editor to update the dependencies. To add the module to an enterprise application, see [Adding modules to an enterprise application](#).

Procedure

1. In the Enterprise Explorer view of the Java EE perspective, right-click your project file and select **Properties**. On the Properties page, select **Deployment Assembly**. If you have a deployment descriptor, you can also open **Deployment Assembly** by clicking the Manage Utility Jars link on the deployment descriptor page.
2. Click **OK**.

Results

Tip: If you need to compile against a server runtime JAR files during development, you do not need to add these JAR files as dependent JAR files. The workbench manages this by using the target server property for the project. The workbench adds the appropriate libraries to your project's build and class path that is based on the target server. For more information, see [Specifying target servers for J2EE projects](#).

- [Adding modules to an enterprise application](#)

You can use the Application Deployment Descriptor editor to add existing projects in your workspace to your enterprise application as new modules.

Parent topic: [Defining Java EE applications](#)

Related concepts:

[Application deployment descriptor editor](#)

Related tasks:

[Defining Java EE applications](#)

Adding modules to an enterprise application

You can use the Application Deployment Descriptor editor to add existing projects in your workspace to your enterprise application as new modules.

About this task

For each module that you add to the enterprise application, a `<module>` element is defined in the deployment descriptor (`application.xml`).

Tip: There are shortcuts for adding modules or utility JAR files to an enterprise application. In the Enterprise Explorer view, drag a project and drop it into:

- The modules box on the **Overview** section of the deployment descriptor for the enterprise application.

Procedure

1. In the Enterprise Explorer view of the Java™ EE perspective, right-click the deployment descriptor for your enterprise application project and select **Open With > Deployment Descriptor Editor** to open the Application Deployment Descriptor editor.
2. In the modules box on the **Overview** section, click **Add > Module**.
3. Select the existing project in your workspace that you want to add as a module to the application.
4. Click **Finish**. The selected module is added to the enterprise application.
5. If the target server specified for the module is different from the target server that is specified for the enterprise application, the Change Target Server dialog opens. Click **Yes** to update the target server for the module to be the same target server that is specified for the enterprise application.

What to do next

To remove a module from the enterprise application, select the module from the list and click **Remove**.

Parent topic: [Specifying dependent JAR files or modules](#)

Exporting and importing binary modules

You can add binary modules into and remove binary modules from your workspace.

Before you begin

Binary modules are Java™ EE modules (for example, EJB, web, connector, application client, and utility jars) that are in binary state (that is, in a single .jar, .war, or .rar file) within the EAR.

About this task

Using binary modules in your workspace allows for better performance, so it is possible for a developer to only load that active component he is working on in source mode (that is, as a project) while leaving all other modules in binary state. This reduces workspace size by requiring projects for only those modules being modified. It improves performance because no builders or validators execute on binary modules.

None of the deployment descriptor files, .java files, mapping files, or the MANIFEST.MF files can be modified; however everything can be read from them in their respective editors in read-only mode.

Procedure

1. To export a module as a binary, follow these steps:

- A. If the application includes EJBs, then you need to ensure that the EAR is prepared for deployment before the export. Right-click the EJB project, and select **Java EE Tools > Prepare for deployment**.
- B. Right-click your project, and select **Export > Shared Ear File**.
- C. On the Shared Ear Export page, click **Browse** to choose a destination folder for the EAR file.
- D. Select **Optimize for a specific server runtime** to optimize server performance, or clear **Optimize for a specific server runtime** if you do not want to optimize performance. Accept the default server runtime, or select a different runtime. **Note:** If you selected WebSphere® Application Server as your runtime, the **Optimize for a specific server runtime** has no effect on your deployment, because this feature is not implemented for WebSphere Application Server.
- E. Select **Overwrite existing file** or clear **Overwrite existing file** if you do not want to overwrite existent files.
- F. Click **Finish**.

2. To import a module as a binary, follow these steps:

- A. Right-click your project, and select **Import > Shared Ear File**.
- B. On the Shared Ear Import page, click **Browse** to locate the EAR file that you want to import.
- C. Select the **Ear project** associated with the Ear file.
- D. Select the **Target runtime** for your imported project.
- E. Click **Finish**.

Parent topic: [Defining Java EE applications](#)

Related concepts:

[Application deployment descriptor editor](#)

Related tasks:

[Defining Java EE applications](#)

Java EE deployment assembly

Deployment assembly property pages allow you to add flexible resource and dependency mapping to your applications.

To access the deployment assembly page, right-click your Java™ EE project, and select **Properties > Deployment Assembly**. For Java EE modules the page consists of two tabs:

- The Deployment Assembly tab
- The Manifest Entries tab

In EAR projects, only the Deployment Assembly tab is visible.

The Deployment Assembly tab

The Deployment Assembly table consists of two columns: the Deploy Path column and the Source column.

- The Deploy Path: The Deploy Path column represents the path where the reference is located within the packaged archive. You can modify this location to customize how you want your packaged archive to be organized. However, removing default folder mappings or modifying their deploy paths should be done carefully, since issues with deployment might be encountered if the changes violate the Java EE specification requirements.
- The Source Column: The Source column represents the location of the resource relative to the project, the file system, or the workspace, depending on which type of dependency was added.

Adding dependencies

- **Archive from Workspace:** If you add an archive from workspace reference, you can then add a reference to an archive file that is inside any project in the workspace. Adding this reference places the chosen archive in the specified deployment path.
- **Archives from File System:** If you add an Archive from File System reference, you can then add a reference to an archive file that is anywhere in the file system. Adding this reference places the chosen archive in the specified deployment path.
- **Java Build Path Entries:** This option is not available for EAR projects. If you add a Java Build Path Entries reference, you can then add a reference to a Java Build Path Entry that is already defined in your project. You can add references to the following types of Java Build Path Entries.
 - You can add references to loose JAR files that are added to a project classpath directly by the Java Build Path property sheet.
 - You can add references to variables that are defined in the workspace preferences. Access these variables by selecting **Window > Preferences > Java > Build Path > Classpath Variables preference page**.
 - You can add references to user libraries that are defined in the workspace preferences. Access these libraries by selecting **Window > Preferences > Java > Build Path > User Libraries preference page**. All archive files that are contained in the library are placed into the specified deployment path. When you add a Java Build Path Entries reference, the deployment path is `WEB-INF/lib` for Web projects. For other modules that are not web modules, the default deployment path is relative to the parent EAR module. If the parent EAR module is version 5 and above, the deployment path is the `lib` directory, such as `../lib`. If the parent EAR module is version 1.4 or below, the deployment path puts the Java Build Path Entry at the same level as the module, such as `../`.
- **Project:** If you add a project reference, you can then bundle the chosen workspace project into an archive, and place the archive in the specified deployment path.
- **Folder:** If you add a folder reference, you can then select any project folder, and map the folder into a deployment path within the archive. By default, the added folder mappings' deployment paths are set to the root folder of the archive.

The Manifest Entries tab

Java EE classpath management

The Java EE development experience in your workspace simulates the runtime environment closely, reducing the possibility of unforeseen issues that may appear after you publish your application. Each module's MANIFEST file is

managed capturing the runtime visibility by duplicating dependent JAR files or other projects on the project classpath. The MANIFEST tab of the deployment assembly captures the existing entries, and allows you to add additional entries within the scope of the parent EAR module. The list of entries available is limited to files of JAR type that reside within the deployed EAR module, and are not included in the designated EAR lib directory. In Java EE 5, the library folder was introduced to EAR modules as a simple technique for sharing JAR files used by other contained modules, but are no longer required to add entries in each respective MANIFEST file. The Deployment Assembly page for EAR projects includes a field for changing the default location of this folder. By default any jar in the `/lib` folder is shared, and your project classpath will include these JAR files automatically. In addition, the JAR file is not required to physically reside in this folder if a mapping is created from its development location to the EAR file's runtime `/lib` folder.

Web modules also have a special folder that allows the sharing of libraries under `WEB-INF/lib`. Similar to the EAR's library folder, any loose JAR file or workspace project mapped to this location will automatically appear on the Web project's classpath.

WebSphere Application Server Loose configuration

The WebSphere® test environment prepares your project for deployment as you are developing, and uses the flat project contents as they are, meanwhile mapping to a standard Java EE runtime structure that WebSphere Application Server can understand. This mapping in WebSphere Application Server is called loose configuration, and allows hot deployment of Java EE applications without the need for special packaging before publish. The new assembly capability allows you to use non-default project layouts, and artifact mappings that might require some packaging at publish time, and could impact publishing performance. In these cases, validation messages warn you of these potential issues. If no messages appear, then your application can run as-is, with no performance penalty.

- [Editing the Java EE deployment assembly page](#)

The Deployment Assembly project property page is shared among all Java EE project types, including OSGi projects.

Parent topic: [Defining Java EE applications](#)

Related tasks:

[Editing the Java EE deployment assembly page](#)

Editing the Java EE deployment assembly page

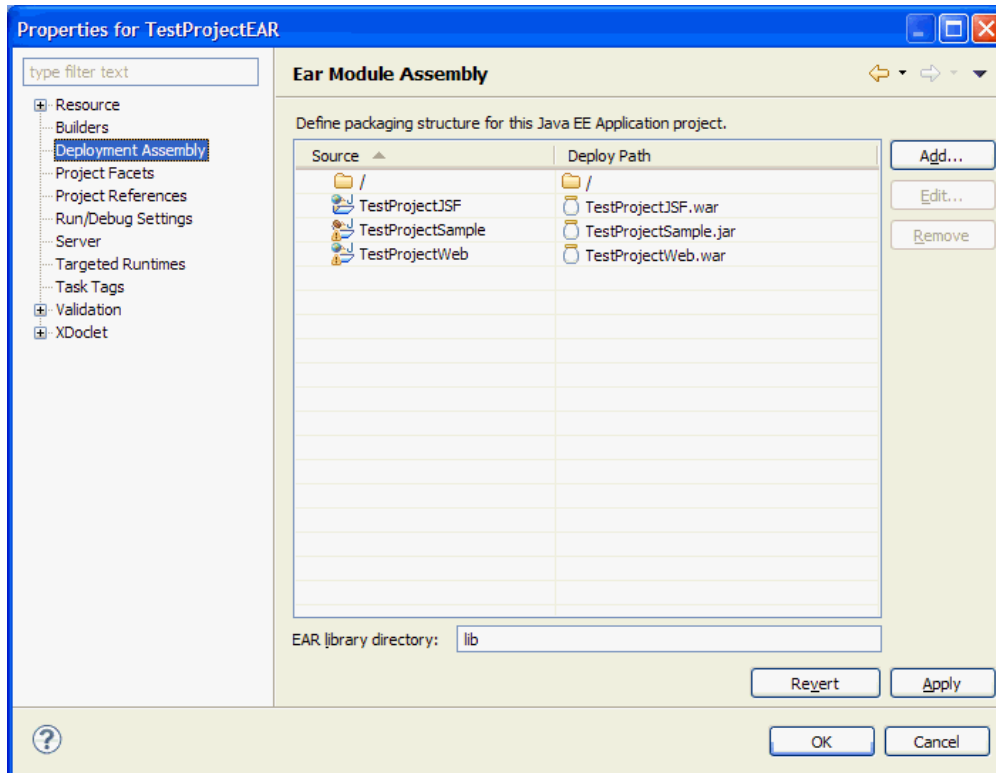
The Deployment Assembly project property page is shared among all Java™ EE project types, including OSGi projects.

Procedure

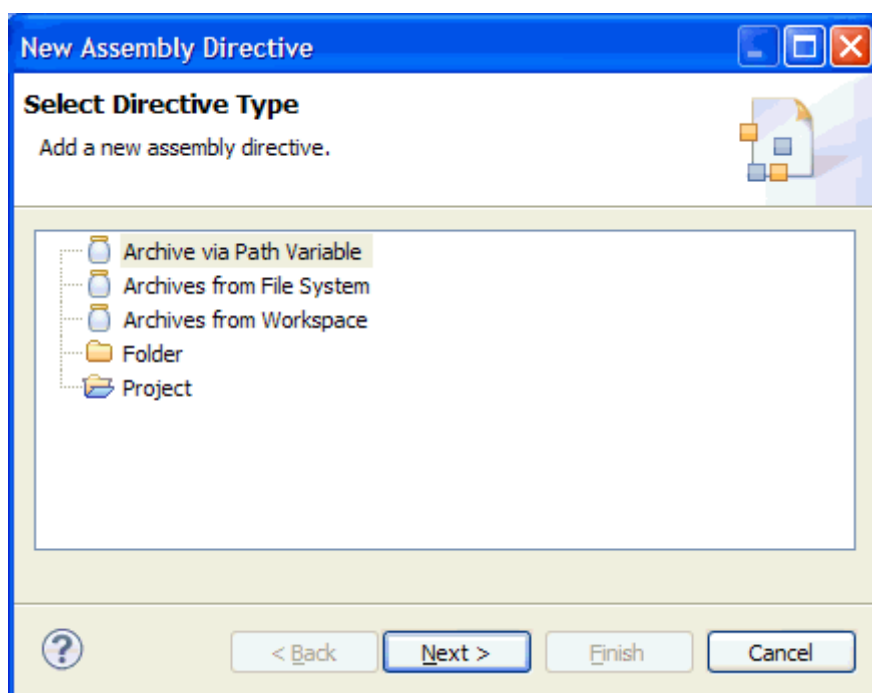
- Editing your EAR project deployment assembly page:

1. To use the Deployment Assembly page, right-click your EAR project and select **Properties > Deployment Assembly**.

The Deployment Assembly page consists of a table that contains a Deploy Path column and a Source column:



2. The Deploy Path column represents the path within the packaged archive. Click **Add** to add mappings:



You can add the following references:

- **Archive via path Variable**
- **Archives from File System**
- **Archives from Workspace**
- **Folder**

- **Project**: enables bundling of the project into an archive, and placed in the runtime location specified. Project references are often used to bundle utility projects into a web project (WEB-INF/lib) location, which is automatically included on the WAR runtime classpath.

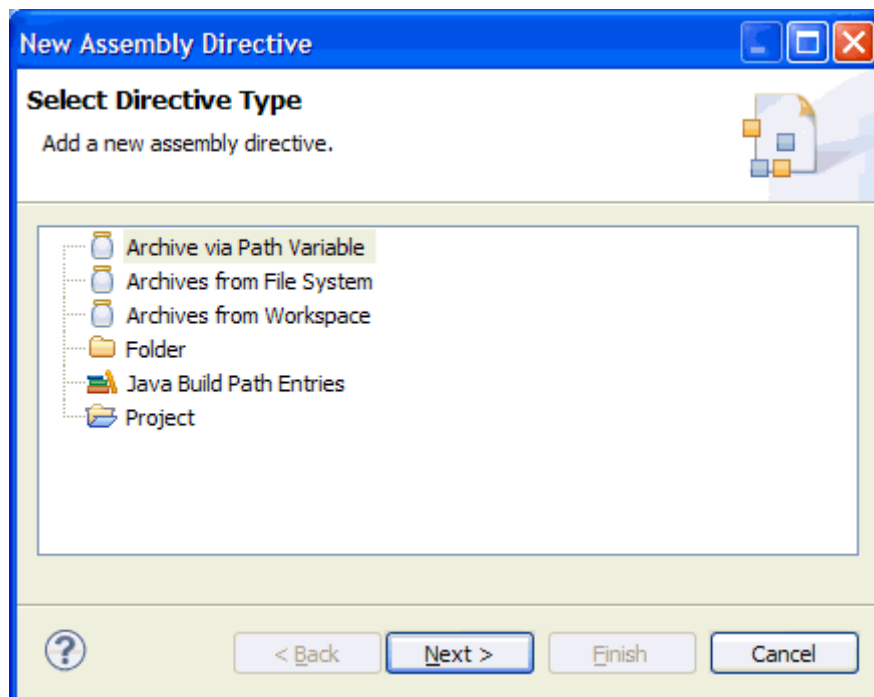
3. The Source column represents the project path. For information about Annotation scanning, see [Excluding files from annotation scanning](#).

- Editing your EJB or web project deployment assembly page:

1. To use the Deployment Assembly page, right-click your Java EE project and select **Properties > Deployment Assembly**. For Java EE modules the page consists of two tabs:

- The Deployment Assembly tab, which consists of a table that contains a Deploy Path column and a Source column.
- The Manifest Entries tab

2. On the Deployment Assembly tab, click **Add** to add mappings:

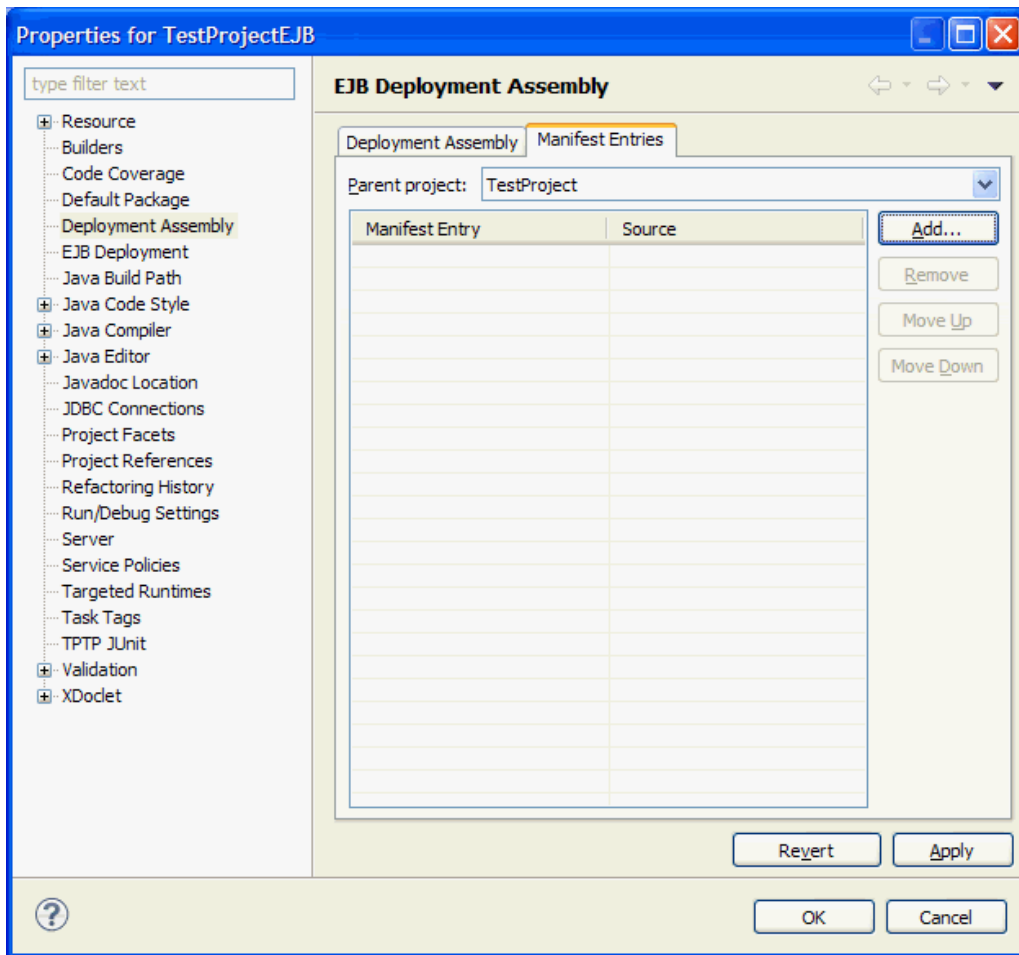


You can add the following references:

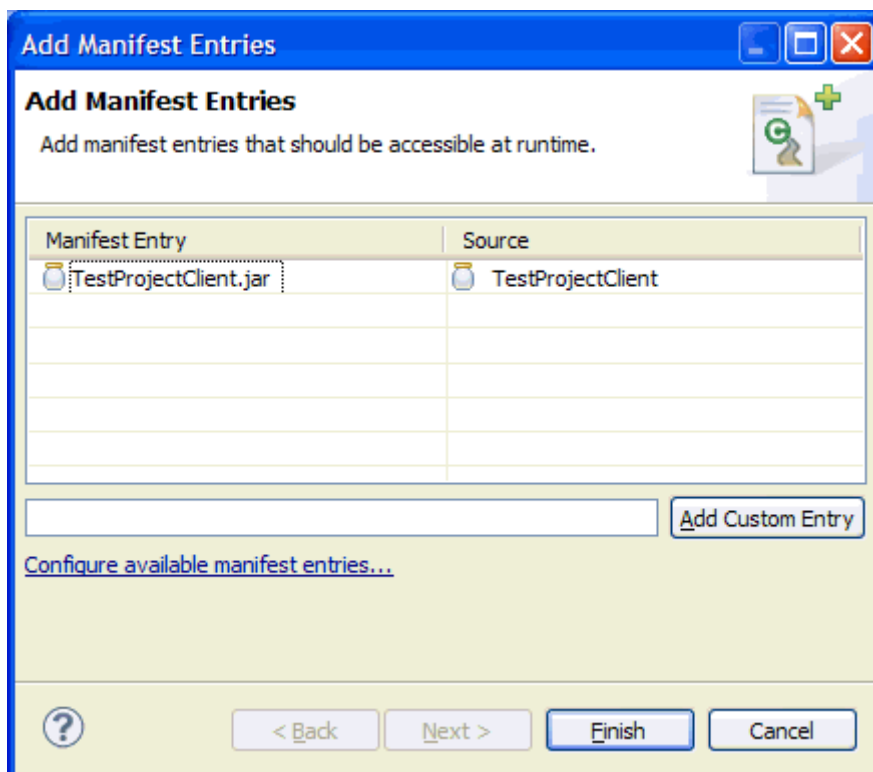
- **Archive via path Variable**
- **Archives from File System**
- **Archives from Workspace**
- **Folder**
- **Java Build Path Entries**

- **Project**: enables bundling of the project into an archive, and placed in the runtime location specified. Project references are often used to bundle utility projects into a web project (WEB-INF/lib) location, which is automatically included on the WAR runtime classpath.

3. On the Manifest Entries tab, click **Add** to add manifest entries:



You can add custom manifest entries or configure available entries:



Parent topic: [Java EE deployment assembly](#)

Related concepts:

[Java EE deployment assembly](#)

Enterprise application security

You can provide security for your Java™ EE enterprise application using annotations or using deployment descriptors.

Security is important in the Java EE environment, and is accomplished through authentication and authorization.

Authentication verifies the identity of a given user, typically by requiring the user to enter a user name and password. In the Java EE environment, authentication is associated with a realm. The realm can store user identity information in many ways, including files, LDAP directories, and even databases that are accessed through JDBC. Authorization grants access control permissions that are based not only on what software is running but also on identity of the authenticated user who is running it. Each time a user logs in, he or she is granted a set of permissions for each application.

Before Java EE 5, if you wanted to use authorization for a given application, you needed to specify authorization information in the application deployment descriptors `ejb-jar.xml` or `web.xml`. One of the main focuses of Java EE is to simplify development of Java EE applications. Starting in Java EE 5, developers can specify annotations in Java source files instead of putting metadata in deployment descriptors. Annotations simplify the development of Java EE applications, shortening development cycles and reducing the total cost of ownership.

You can secure your enterprise application using annotations, or, if you prefer, using deployment descriptions. For a web module, you still need to specify a `<security-constraint>` in the `web.xml` application deployment descriptor in order to have authorization constraints, just as you did in J2EE 1.4. In the Java EE 5 environment, the permissions-related annotations are only defined for EJB modules. For EJB security, see [Securing EJB's](#)

- Annotations to secure Java EE applications

You can provide security for your Java EE enterprise application directly in your source code using annotations.

- EJB Security

You can provide security for your EJB application using annotations or using deployment descriptors.

Parent topic: [Developing Java EE Applications](#)

Parent topic: [Developing EJB 3.x applications](#)

Related concepts:

[Learn about Java EE Applications](#)

[Java EE: Overview](#)

[Tools for Java EE development](#)

[Project facets](#)

[Java EE with annotations overview](#)

[Defining Java EE applications](#)

[Annotations to secure Java EE applications](#)

Related tasks:

[Developing Java EE Applications](#)

[Setting Java EE preferences](#)

[Creating and configuring Java EE projects using wizards](#)

[Validating code in enterprise applications](#)

Annotations to secure Java EE applications

You can provide security for your Java™ EE enterprise application directly in your source code using annotations.

Common security annotations: JSR 250 defines a number of common security annotations. Five security annotations are defined:

- **javax.annotation.security.PermitAll:**

- Can be used at type or method level.
- Indicates that the given method or all business methods of the given EJB are accessible by everyone.

- **javax.annotation.security.DenyAll:**

- Can be used at method level.
- Indicates that the given method in the EJB cannot be accessed by anyone.

- **javax.annotation.security.RolesAllowed:**

- Can be used at type or method level.
- Indicates that the given method or all business methods in the EJB can be accessed by users that are associated with the list of roles.

- **javax.annotation.security.DeclareRoles:**

- Can be used at type level.
- Defines roles for security checking. To be used by `EJBContext.isCallerInRole`, `HttpServletRequest.isUserInRole`, and `WebServiceContext.isUserInRole`.

- **javax.annotation.security.RunAs:**

- Can be used at type level.
- Specifies the RunAs role for the given components.

Using security annotations

- For the annotations `@PermitAll`, `@DenyAll`, and `@RolesAllowed`, a class-level annotation applies to a class and a method-level annotation applies to a method. Method-level annotation overrides the behavior of class level annotation.

```
@Stateless
@RolesAllowed("team")
public class TestEJB implements Test {
    @PermitAll
    public String hello(String msg) {
        return "Hello, " + msg;
    }

    public String goodbye(String msg) {
        return "Goodbye, " + msg;
    }
}
```

In this example, the `hello()` method is accessible by everyone, and the `goodbye()` method is accessible by users of role `team`.

- The `@DeclareRoles` annotation defines a list of roles to be used by a given component. In the Java EE 5 or 6 environment, you can look up the resource by using the `@javax.annotation.Resource` and verify whether the user is of the given role by invoking the following APIs: *Table 1. . API used by Java EE component*

Component	API used to check role
EJB	<code>javax.ejb.EJBContext.isCallerInRole(role)</code>

Servlet	<code>javax.servlet.http.HttpServletRequest.isUserInRole(role)</code>
Web Service	<code>javax.xml.ws.WebServiceContext.isUserInRole(role)</code>

- The `@PermitAll`, `@DenyAll`, and `@RolesAllowed` annotations allow you to implement most of the authorization decision. However, to accomplish more complicated logic, use the `@DeclareRoles` annotation. For example, suppose the hello method is to be accessible by a user who is in role A and who is not in role B at the same time. The following code clip achieves this goal: `@Stateless`

```

@DeclareRoles({"A", "B"})
public class TestEJB implements Test {
    @Resource private SessionContext sc;
    public String hello(String msg) {
        if (sc.isCallerInRole("A") && !sc.isCallerInRole("B")) {
            ...
        } else {
            ...
        }
    }
}

```

Invalid use of security annotations

- More than one of `@DenyAll`, `@PermitAll`, `@RolesAllowed` cannot apply to the same method. For example, the following usage is not valid: `@PermitAll`

```

@DenyAll
public String test()

```

- Two `@RolesAllowed` annotations cannot apply to the same methods. For example, the following usage is not valid and fails in compilation: `@RolesAllowed("team")`

```

@RolesAllowed("otherteam")
public String hello()

```

Instead, use this structure: `@RolesAllowed({"team", "otherteam"})`

```

public String hello()

```

Parent topic: [Enterprise application security](#)

Related concepts:

[Enterprise application security](#)

8.5.5.6 Setting permissions in the Java EE Application Permission Editor

After you create your `permissions.xml` file, you can edit this file with the Java EE Application Permission Editor to set rights and permissions for your Java™ EE applications.

Before you begin

Java EE 7 is required to use the Java EE Application Permission Editor.

Create the `permissions.xml` file if it does not exist.

1. Go to **File > New > Other**.
2. Select **Java EE Application Permission file** from the Java EE Category section.
3. Click **Next**.
4. Select your project from the drop-down menu. Only valid projects are shown.
5. Click **Finish**.

About this task

Set permissions for your applications by editing your `permissions.xml` file with the Java EE Application Permission Editor.

Procedure

1. Open the `permissions.xml` file with the Java EE Application Permission Editor.
2. To add permissions, click **Add...**
3. Select **Permission** and click **OK**.
4. Select the class name that you want to add from the drop-down menu, and then click **OK**. The Details section of the Java EE Application Permission Editor is displayed.
Important: If you select **Custom Permission...** from the drop-down menu, then you must manually enter a value for the **Class name**, **Name**, and **Actions** fields. You cannot use **Suggested Values** to automatically populate the **Name** and **Actions** fields for permissions that are not in the drop-down menu.
5. Optional: To change or add the name value for the permission, either enter it in the **Name** field, or click the **Suggested Values** button for the **Name** field to see a list of possible values.
 - A. Select a name value from the **Values** drop-down menu.
 - B. Click **OK**.**Tip:** Some classes do not have any available name values. The **Name** field can be left blank.
6. Optional: To change or add the actions value for the permission, either enter it in the **Actions** field, or click the **Suggested Values** button for the **Actions** field to see a list of possible values.
 - A. Select one or more actions values from the list.
 - B. Click **OK**.**Tip:** Some classes do not have any available actions. The **Actions** field can be left blank.
7. Save the `permissions.xml` file.

Parent topic: [Developing Java EE Applications](#)

Validating code in enterprise applications

The workbench includes validators that check certain files in your enterprise application module projects for errors.

About this task

By default, the workbench validates your files automatically after any build, including automatic builds. You can also begin the validation process manually without building.

On the workbench Preferences window, you can enable or disable validators to be used on your projects. Also, you can enable or disable validators for each enterprise application module project individually on the Properties page for that project.

Each validator can apply to certain types of files, certain project natures, and certain project facets. When a validator applies to a project facet or nature, the workbench uses that validator only on projects that have that facet or nature.

Likewise, most validators apply only to certain types of files, so the workbench uses those validators only on those types of files.

Procedure

1. Click **Window > Preferences > Validation**. The Validation page of the Preferences window lists the validators available in your project and their settings.
2. Optional: Review the check box options to customize your validation settings:

Option	Description
Allow projects to override these preference settings	Select to set individual validation settings for one or more of your projects.
Suspend all validators	Select to prevent validation at the global level.
Save all modified resources automatically prior to validating	Select to save resources you have modified before the validation begins.
Show a confirmation dialog when performing manual validation	Select to show an informational dialog after a manual validation request has completed.

3. In the list of validators, select the check boxes next to each validator you want to use at the global level. Each validator has a check box to specify whether it is used on manual validation and/or on build validation.
4. Tune a validator by clicking the button in the **Settings** column. Not all validators have more settings.
5. Begin the validation process by one of the following methods:
 - Right-click a project and click **Validate**.
 - Start a build.

Results

Any errors found by the validators are listed in the Problems view. If you want to set individual validation settings for one or more of your projects, see [Overriding global validation preferences](#) for more information.

What to do next

- **Validator tuning**

Whether or not a validator validates a particular resource depends on the filters that are in place for that validator.

- **Validating Java EE projects**

The workbench includes validators that check certain files in your Java EE enterprise application projects for errors.

- **Manually validating code**

When you run a manual validation, all resources in the selected project are validated according to the validation settings.

- **Common validation errors and solutions**

This table lists the common error messages that you might encounter when you validate your projects.

- **Disabling validation**

You can disable one or more validators individually or disable validation entirely. Also, you can set validation settings for your entire workspace and for individual projects.

- **Selecting code validators**

You can select specific validators to run during manual and build code validation. You can set each validator to run on manual validation, build validation, both, or neither.

- **Overriding global validation preferences**

For a given project, you can override the global validation preferences.

Parent topic: [Developing Java EE Applications](#)

Validator tuning

Whether or not a validator validates a particular resource depends on the filters that are in place for that validator.

When a validator is first developed, the implementer of the validator defines a default set of filters. These filters might be based on:

- file extensions
- folder or file names
- project natures
- project facets
- content types

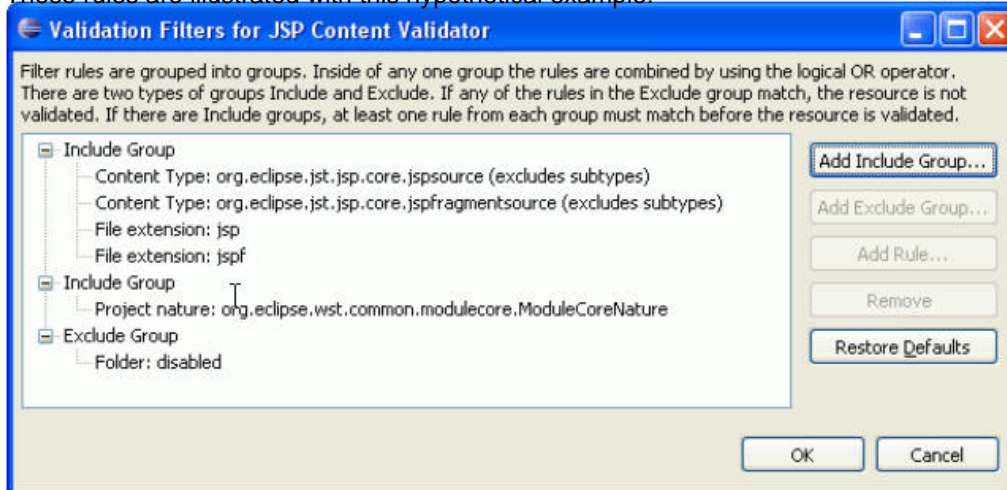
Through the Validation Filters dialog, you are able to further tune these settings. Normally you would keep the defaults; however, two reasons why you might want to tune validation are as follows:

- Performance: if you have a large workspace, you might reduce the amount of validation.
- Non-standard conventions: if you use a non-standard naming convention (for example stores XML in files with an .acme.xml extension), you could still enable the appropriate validators to run against those files.

You can access this dialog by clicking **Window > Preferences > Validation** and then clicking **Settings** next to each validator.

Filters are stored in groups. There are two types of groups: Include groups and Exclude groups. You can have as many Include groups as you like. Filters inside of an Include group cause resources to be validated. If any rule matches, then the entire group matches. Inside of a group the filter rules are OR'd together. However, individual Include groups are AND'ed together. You can have one Exclude group. If any of its filter rules match, then the resource is excluded. Exclusion takes precedence over inclusion.

These rules are illustrated with this hypothetical example:



- If the resource is in the disabled folder, it will be excluded because exclusion takes precedence over everything else.
- If the resource does not have the JSP source content type, and it does not have the JSP fragment source content type, and it does not have a file extension of .jsp or .jspx then it is excluded because none of the rules in the first group matched.
- If the project does not have the module core nature then, it is excluded because the single rule in the second group did not match.
- Otherwise, the resource is validated by this particular validator.

To add a rule to a group, select the group and click **Add Rule**.

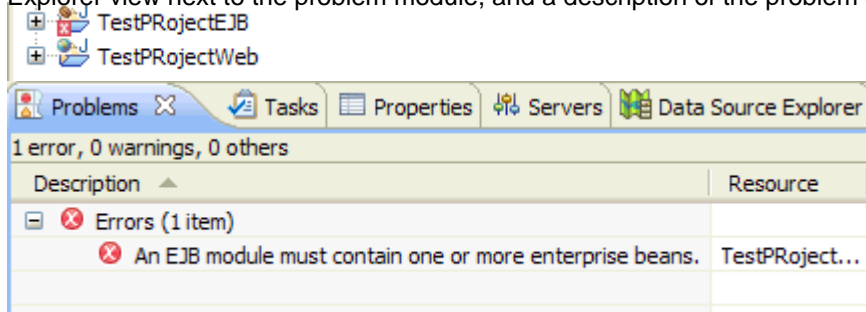
Parent topic: [Validating code in enterprise applications](#)

Validating Java EE projects

The workbench includes validators that check certain files in your Java™ EE enterprise application projects for errors.

Procedure

After you create a Java EE project, the Java EE validators check the code. A red error mark appears in the Enterprise Explorer view next to the problem module, and a description of the problem or error appear in the Problems view:



Parent topic: [Validating code in enterprise applications](#)

Manually validating code

When you run a manual validation, all resources in the selected project are validated according to the validation settings.

About this task

The validators that are used depend on the global and project validation settings. When you validate a project manually, the global settings are used unless both of the following are true:

- The **Allow projects to override these preference settings** check box is selected on the global validation preferences page.
- The **Enable project specific settings** check box is selected on the project's validation preferences page.

Whether the workbench uses the global or project validation preferences, only the validators that are selected to run on manual validation are used when you run a manual validation.

Procedure

1. Select the project that you want to validate.
2. Right-click the project and then click **Validate**. If this option is not available, validation is disabled or there are no validators enabled for the project. To enable validation at the global level, see [Validating code in enterprise applications](#).
To enable validators for this project, see [Overriding global validation preferences](#).

Results

The workbench validates the project using the enabled validators. Any errors that are found by the validators are listed in the Problems view. **Note:** The errors that are listed in the validation results dialog do not necessarily correspond one for one with the errors listed in the problems view. The validation results display all of the errors found, whereas duplicate errors appear only once in the Problems view.

What to do next

Parent topic: [Validating code in enterprise applications](#)

Common validation errors and solutions

This table lists the common error messages that you might encounter when you validate your projects.

Message prefix	Message	Explanation
Application Client validator		
CHKJ1000	Validation failed because the application client file is not valid. Ensure that the deployment descriptor is valid.	The application-client.xml file cannot be loaded. The project metadata cannot be initialized from the application-client.xml file. Ensure the following: that the META-INF folder exists in the application client project that META-INF contains the application-client.xml file that META-INF is in the project's classpath. Validate the syntax of the application-client.xml file: in the Navigator view, highlight the application-client.xml file, right-click, and select Validate XML file . If both 1) and 2) are okay, close the project, reopen the project, and rebuild the project. The project metadata refreshes.
EAR validator		
CHKJ1001	The EAR project {0} is invalid.	The application.xml file cannot be loaded. The project metadata cannot be initialized from the application.xml file. Ensure the following: that the META-INF folder exists in the EAR project that META-INF contains application.xml that META-INF is in the project's classpath. Validate the syntax of the application.xml file: in the Navigator view, highlight the application.xml file, right-click, and select Validate XML file . If both 1) and 2) are okay, close the project, reopen the project, and rebuild the project. The project metadata refreshes.
EJB validator		
CHKJ2102	Either a finder descriptor, or a matching custom finder method on the {0} class, must be defined.	A finder descriptor must exist for every finder method.

CHKJ2875E	<ejb-client-jar> {0} must exist in every EAR file that contains this EJB module.	If <ejb-client-jar> is specified in <code>ejb-jar.xml</code> , a corresponding EJB client project must contain the home and remote interfaces and any other types that a client will need. If these types are all contained in a single EJB project, delete the <ejb-client-jar> line in the deployment descriptor. Otherwise, ensure that the EJB client project exists, is open, and is a project utility JAR in every EAR that uses this EJB project as a module.
CHKJ2905	The EJB validator did not run because <code>ejb-jar.xml</code> could not be loaded. Run the XML validator for more information.	CHKJ2905 means that the project's metadata could not be initialized from <code>ejb-jar.xml</code> . Ensure the following: that the META-INF folder exists in the EJB project that META-INF contains <code>ejb-jar.xml</code> that META-INF is in the project's classpath. Validate the syntax of the <code>ejb-jar.xml</code> file: in the Navigator view, highlight the <code>ejb-jar.xml</code> file, right-click, and select Validate XML file . If both 1) and 2) are okay, close the project, reopen the project, and rebuild the project. The project metadata refreshes.
JSP validator		
IWAW0482	No valid JspTranslator	There is a path problem with the project; the JSP Validator needs access to the WAS runtime code. If IWAW0482E appears on all web projects, check the Variable or JRE path: Check the global preferences (Window > Preferences > Java > Installed JREs) and make sure that the location for the JRE is pointing to a valid JRE directory. Ensure that the classpath variables (Window > Preferences > Java > Classpath Variables) are set correctly.
WAR validator		

CHKJ3008	Missing or invalid WAR file.	The <code>web.xml</code> file cannot be loaded. The project metadata cannot be initialized from the <code>web.xml</code> file. Ensure the following: that the WEB-INF folder exists in the web project that WEB-INF contains the <code>web.xml</code> file that WEB-INF is in the project's classpath. Validate the syntax of the <code>web.xml</code> file: in the Navigator view, highlight the <code>web.xml</code> file, right-click, and select Validate XML file . If both 1) and 2) are okay, close the project, reopen the project, and rebuild the project. The project metadata refreshes.
XML validator		
	The content of element type <code>ejb-jar</code> is incomplete, it must <code>match(description?, display-name?, small-icon?, large-icon?, enterprise-beans, assembly-descriptor?, ejb-client-jar?)</code> .	The EJB 1.1 and 2.0 specifications mandate that at least one enterprise bean must exist in an EJB <code>.jar</code> file. This error message is normal during development of EJB <code>.jar</code> files and can be ignored until you perform a production action, such as exporting or deploying code. Define at least one enterprise bean in the project.

Parent topic: [Validating code in enterprise applications](#)

Disabling validation

You can disable one or more validators individually or disable validation entirely. Also, you can set validation settings for your entire workspace and for individual projects.

About this task

To disable validators in your project or workspace, complete the following steps:

Procedure

1. Click **Window > Preferences** and select **Validation** from the navigation pane. The Validation page of the Preferences window lists the validators available in your project and their settings.
2. To disable individual validators, clear the check boxes next to each validator that you want to disable. Each validator has a check box to specify whether it is enabled for manual validation or on a build.
3. Optional: You can also change the following check box options on this page:

Option	Description
Allow projects to override these preference settings	Select to set individual validation settings for one or more of your projects.
Suspend all validators	Select to prevent validation at the global level. If you select this check box, you can still enable validation at the project level.

4. Click **OK**.

Results

If you want to set individual validation settings for one or more of your projects, see [Overriding global validation preferences](#)

What to do next

Parent topic: [Validating code in enterprise applications](#)

Selecting code validators

You can select specific validators to run during manual and build code validation. You can set each validator to run on manual validation, build validation, both, or neither.

About this task

To choose the validators that you want to use for a project, complete the following steps:

Procedure

1. Click **Window > Preferences** and select **Validation** in the navigation pane. The Validation page of the Preferences window lists the validators available in your project and their settings.
2. Clear the **Suspend all validators** check box.
3. Optional: If you want to set individual validation settings for one or more of your projects, select the **Allow projects to override these preference settings** check box.
4. In the list of validators, select the check boxes next to each validator you want to use at the global level. Each validator has a check box to specify whether it is used on manual validation or on a build. **Note:** If you deselect any validator that is selected, any messages that are associated with the deselected validator will be removed from the task list, the next time you perform a full build.
5. Tune a validator by clicking the button in the **Settings** column. Not all validators have additional settings.

Results

If you want to set individual validation settings for one or more of your projects, see [Overriding global validation preferences](#). Like the global validation preferences, you can set validators at the project level to run on manual validation, build validation, both, or neither.

What to do next

Parent topic: [Validating code in enterprise applications](#)

Overriding global validation preferences

For a given project, you can override the global validation preferences.

About this task

The default validation preferences are specified globally on the Validation page of the Preferences dialog.

Procedure

1. Click **Window > Preferences** and select **Validation** in the navigation pane. The Validation page of the Preferences window lists the validators available in your project and their settings.
2. Select the **Allow projects to override these preference settings** check box and click **OK**. Now individual projects can override the global preferences.
3. Right-click a project and then click **Properties**.
4. In the navigation pane of the Properties window, click **Validation**.
5. Select the **Enable project specific settings** check box. If you did not check the **Allow projects to override these preference settings** check box on the workbench validation preferences page, you will get a warning message since you cannot override workspace validation preferences.
6. To prevent validation for this project, select the **Suspend all validators** check box.
7. In the list of validators, select the check boxes next to each validator you want to use. Each validator has a check box to specify whether it is used on manual validation or on a build.
8. Choose an alternate implementation for a validator by clicking the button in the **Settings** column, then click **OK**. Not all validators have alternate implementations.

What to do next

Parent topic: [Validating code in enterprise applications](#)

Developing EJB 3.x applications

You can create Enterprise Java™Beans (EJB) 3.x applications with the Java EE specification more simply than previous EJB specifications by using annotations. You can edit, provide security for, and test your EJB 3.x applications.

- **EJB 3.x overview**

You can use the workbench to develop and test enterprise beans that conform to the distributed component architecture defined in the Enterprise JavaBeans™ (EJB) 3.2 specification. EJB 3.2, 3.1, and 3.0 are supported.

- **EJB modules**

Enterprise Java bean (EJB) modules are used to assemble one or more enterprise beans into a single deployable unit. An EJB module is stored in a standard Java archive (JAR) file.

- **Creating enterprise beans**

You can use wizards or annotations to create enterprise beans to add to your Java or Enterprise Java Beans (EJB) project. You can create entity beans, session beans, and message-driven beans, and place them in an EJB container.

- **Editing EJB 3.x applications**

You can edit your EJB 3.x applications in your source code by directly modifying your source code and the annotations in the Java editor or by using the Annotations view.

- **Enterprise application security**

You can provide security for your Java EE enterprise application using annotations or using deployment descriptors.

- **Testing EJB 3.x applications**

After you have created an EJB 3.x applications, you can create a Servlet or JSF to test the EJB 3.x applications.

Parent topic:[EJB 3.x overview](#)

EJB 3.x overview

8.5.5.6 You can use the workbench to develop and test enterprise beans that conform to the distributed component architecture defined in the Enterprise JavaBeans™ (EJB) 3.2 specification. EJB 3.2, 3.1, and 3.0 are supported.

Using the EJB 3.x specification, you can develop beans more simply than in the 2.1 standard. You can annotate your Java™™ source code to provide information that was previously contained in XML deployment descriptors. By using Java annotations, you can create EJBs and Java Persistence Architecture beans quickly and easily from plain old Java objects (POJOs). EJBs can be created without implementing EnterpriseBean interfaces.

This product supports the Enterprise JavaBeans™ 3.x. specification levels, and offers minimal assembly tools for 2.x specification levels. All the EJB tools in the product are accessible from the Java EE perspective in the workbench. You can create session beans (stateful or stateless) or message-driven beans by simply specifying the component defining annotation in your Java class. More configuration for your bean can be done by specifying more annotations in the Java class. The Java editor provides validation, content assistance, and QuickFixes for your EJB 3.x annotations, and support for refactoring beans. For richer assistance with the EJB 3.x annotations, you can use the Annotations view to add or delete annotations, and to modify the attribute values of annotations. Deployment descriptors for your EJB 3.x modules are optional, but can be created for extra configuration.

8.5.5.6 For more information about EJB 3.2, see the official specification: [JSR 345: Enterprise JavaBeans 3.2](#) .

Java EE specification makes the creation of EJB 3.x applications simpler than previous EJB specifications. Java EE streamlines EJB development in the following ways:

- **Fewer required classes and interfaces**

- Home and object interfaces are no longer required – you need the business interface only
- No need to implement `javax.ejb.SessionBean`
- No need to declare checked exceptions

- **Optional deployment descriptors**

- Annotations provide component definition and dependency injection

- **Simple lookups**

- new `EJBContext()` interface method replaces JNDI calls

- **Lightweight persistence for object-relational mapping**

- Entities are POJOs that provide an object-oriented view of the data stored in relational database

- **Interceptors**

- Interceptors are objects that can intercept a call to a business method (to handle security for example)
- Similar in purpose and action to Servlet filter or Web services handler
- Provide limited form of aspect-oriented programming

- **Developing EJB 3.x applications**

You can create Enterprise JavaBeans (EJB) 3.x applications with the Java EE specification more simply than previous EJB specifications by using annotations. You can edit, provide security for, and test your EJB 3.x applications.

Parent topic: [Developing enterprise applications](#)

Parent topic: [Developing EJB 3.x applications](#)

EJB modules

Enterprise Java™ bean (EJB) modules are used to assemble one or more enterprise beans into a single deployable unit.

An EJB module is stored in a standard Java archive (JAR) file.

Overview

An EJB module can be used as a standalone module, or it can be combined with other modules to create an enterprise module. An EJB module is installed and run in an enterprise bean container.

An EJB module has the following characteristics:

- It contains one or more deployable enterprise beans.
- Optional (in EJB 3.x): It might contain a deployment descriptor, stored in an Extensible Markup Language (XML) file. This file declares the contents of the module, defines the structure and external dependencies of the beans in the module, and describes how the enterprise beans are to be used at run time.
- It targets one of the these servers:
 - IBM® WebSphere® Application Server version 8.5 (including Liberty profiles).
 - IBM WebSphere Application Server version 8.0.
 - IBM WebSphere Application Server version 7.0, which is already enabled with EJB 3.0 support.

When you create a new EJB project in an enterprise archive (EAR) file, an EJB client project is created by default. EJB client projects allow flexible packaging of the EJB interface classes for use with local and remote client applications.

Note: EJB modules that contain EJB 3.x beans must be at the EJB 3.x facet level when running on the product. To set the EJB module to support EJB 3.x beans, you can set the EJB version in the project facet to 3.x, or you can make sure that the module does not contain an `ejb-jar.xml` deployment descriptor. If the module level is EJB 2.1 or earlier, no EJB 3.x functions, including annotation scanning or resource injection, is performed at runtime.

Deployment

You can deploy an EJB module as a stand-alone application, or combine it with other EJB modules or with Web modules to create a Java application. An EJB module is installed and run in an enterprise bean container.

- An EJB project must be referenced by an enterprise module project (defined as a module in an EAR file) in order to be deployed successfully and run on a server.
- [8.5.5.6](#) Liberty profile servers can deploy standalone EJB modules.

Parent topic: [Developing EJB 3.x applications](#)

Creating enterprise beans

You can use wizards or annotations to create enterprise beans to add to your Java or Enterprise Java Beans (EJB) project. You can create entity beans, session beans, and message-driven beans, and place them in an EJB container.

- [Enterprise beans overview](#)

After you have created your Java or EJB project, you can create session beans, entity beans, and message-driven beans to add to your project.

- [Creating enterprise beans using wizards](#)

After you have created your EJB project, you can use wizards to create session beans, message-driven beans, and JPA entities to add to your project.

- [Creating enterprise beans using annotations](#)

After you have created your Java or EJB project, you can create session beans, message-driven beans, and JPA entities to add to your project.

- [Importing class files to Web projects](#)

Parent topic: [Developing EJB 3.x applications](#)

Enterprise beans overview

After you have created your Java™ or EJB project, you can create session beans, entity beans, and message-driven beans to add to your project.

Enterprise beans

An enterprise bean is a Java component that can be combined with other resources to create Java applications. There are three types of enterprise beans: entity beans, session beans, and message-driven beans. All beans reside in Enterprise Java beans (EJB) containers, which provide an interface between the beans and the application server on which they reside.

You are also able create EJB 3.x beans in Web 3.x projects.

Component-defining annotations

Using component-defining annotations, you can create the following types of enterprise beans: session beans, message-driven beans, and JPA entities. Including the component-defining annotation `@Stateful`, `@Stateless` indicates that the class is a session bean class; including the component-defining annotation `@Singleton` indicates that the class is a singleton class; and including the component-defining annotation `@MessageDriven` indicates that the class is a Message-driven bean class; and including the component-defining annotation `@Entity` indicates that the class is a JPA entity.

- **Session beans:** At a minimum, a session bean developed with the EJB 3.x specification requires a bean class.

1. **Stateful:** A stateful session bean maintains client-specific session information, or conversational state, across multiple method calls and transactions. An instance of a stateful session bean has a unique identity that is assigned by the container at create time.
2. **Stateless:** A stateless session bean does not maintain conversational state. Instances of a stateless session bean have no conversational state. Because a stateless session EJB does not maintain a conversational state, all the data exchanged between the client and the EJB must be passed either as input parameters, or as return value, declared on the EJB business method interface. All instances of a stateless session bean have the same object identifier, which is assigned by the container.
3. **Singleton:** Singleton session beans, new in EJB 3.1, is a new kind of session bean that is guaranteed to be instantiated once for an application in a particular Java Virtual Machine (JVM). Singletons offer similar functionality to stateless session beans but differ from them in that there is only one singleton session bean per application, as opposed to a pool of stateless session beans, any of which may respond to a client request. Like stateless session beans, singleton session beans can implement web service endpoints. Singleton session beans maintain their state between client invocations but are not required to maintain their state across server crashes or shutdowns.

- **Message-driven beans:** Message-driven beans were introduced in EJB 2.0 to support the processing of asynchronous messages from a Java Message Service (JMS). The EJB 2.1 specification expands the definition of the message-driven bean so that it can support any messaging system, not just JMS. In simplest terms, a message-driven bean is a message consumer that can be called by its container. They are invoked by the container when a message arrives. Message beans are another interaction mechanism for invoking EJBs, but unlike session beans, the container is responsible for invoking them when a message is received, not a client (or another bean).

- **Entities using Java Persistence API (JPA):** Entities use the new Java Persistence API that is part of the Java EE 5 platform. Unlike EJB components that use container-managed persistence (CMP), entity objects that use the new APIs are no longer components, but are simply Java objects. This makes entities more lightweight and the programming model simpler to use. For more information about JPA, see [JPA documentation](#).

Guidelines for developing EJBs

While EJB 3.x provides a flexible and simple programming model, here are a few of the suggested rules for developing EJBs:

- Each entity must be a POJO and the class must be concrete (therefore neither abstract or final).

- The class must have a no-argument constructor; if none is present, the compiler adds a default constructor
- The POJO must implement at least one POJI (plain old Java interface); you do not need to include an interface, but you can include different interfaces for local and remote clients.
- If the business interface includes an @Remote annotation, all the parameters declared on the interface must implement java.io.Serializable.
- A session EJB can subclass a POJO, but cannot subclass another session EJB.

You can create enterprise beans in one of the following ways:

- Create new enterprise beans using wizards.
- Create new enterprise beans using Java EE annotations.
- Import enterprise beans from EJB JAR files.

Parent topic: [Creating enterprise beans](#)

Creating enterprise beans using wizards

After you have created your EJB project, you can use wizards to create session beans, message-driven beans, and JPA entities to add to your project.

Before you begin

You need to have an EJB project created in your workspace. You are also able to create EJB 3.0 and 3.1 beans in Web 3.0 projects.

Procedure

In the Java™ EE perspective, right-click your EJB Project and select **New > Session Bean** OR **New > Message-Driven Bean**. Click **Next**

- **Creating a stateless session bean using wizards**

You can use Create EJB 3.x Session Bean wizard to create a stateless session bean and add it to your project.

- **Creating a session bean with JPA entities**

You can use the Create EJB 3.x Session Bean wizard to create a session bean and a JPA entity in your EJB project.

- **Creating a stateful session bean using wizards**

You can use the Create EJB 3.x Session Bean wizard to create a stateful session bean and add it to your project.

- **Creating an EJB timer by using wizards**

You can use the Create EJB Timer wizard to create Enterprise JavaBeans (EJB) Timers and add them to your project.

- **Creating a message-driven bean using wizards**

You can use the Create EJB 3.1 Message-Driven Bean wizard to create a message-driven bean and add it to your project.

- **Packaging EJB content in web application archive (WAR) modules**

EJB functionality that is supported for beans packaged inside EJB JAR modules is also supported for beans packaged inside WAR modules. A bean packaged inside a WAR module is capable of the same behavior as a bean packaged inside an EJB JAR module.

- **Deleting EJB 3.x beans**

Once you have created an Enterprise Java bean, you might want to delete it. The instructions in this document describe the basic steps that you can take to delete the bean.

Parent topic: [Creating enterprise beans](#)

Creating a stateless session bean using wizards

You can use Create EJB 3.x Session Bean wizard to create a stateless session bean and add it to your project.

Before you begin

You must have a Java™ project, an EJB project, or a web project created in your workspace.

Procedure

1. In the Java EE perspective, right-click your project, and select **New** > **Session Bean**. The Create EJB 3.x Session Bean wizard appears.
2. In the **Source folder** field, select the source folder for the new bean.
3. In the **Java package** field, type the package name for the new bean.
4. In the **Bean name** field, type the name that you want to assign to the enterprise bean. By convention, bean names begin with an uppercase letter. **Note:** You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes associated with enterprise beans.
5. Select Remote to add a remote interface and select Local to add a local interface, and click **Finish**.
6. Select **Asynchronous**. By doing so, you are marking all the business methods of the session bean as asynchronous methods. This action adds the `@Asynchronous` annotation to the class.

8.5.5.6 This capability is present beginning with Luna and with Mars. It is also present for the EJB 3.1 and EJB 3.2 projects.

7. In the Java class editor, underneath the package declaration, you can see the `@stateless` annotation. Your class also contains reference to Local and Remote interfaces, if you selected to create them:

```
import javax.ejb.Stateless;

/**
 * Session Bean implementation class TestBean
 */
@Stateless
public class TestBean implements TestBeanRemote, TestBeanLocal {

    /**
     * Default constructor.
     */
    public TestBean() {
        // TODO Auto-generated constructor stub
    }
}
```

8. Define the client views and interfaces. For EJB 3.0 or later beans, you can include a remote client interface, a local interface, or both. Here is an example of a simple Remote interface:

```
import javax.ejb.Remote;

@Remote
public interface RemoteCounter {
    public int increment();
}
```

```
public int getTheValue();  
}
```

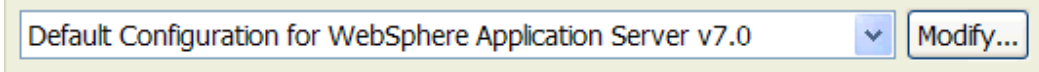
Parent topic:[Creating enterprise beans using wizards](#)

Creating a session bean with JPA entities

You can use the Create EJB 3.x Session Bean wizard to create a session bean and a JPA entity in your EJB project.

Procedure

1. Follow the steps for creating an EJB project. After the **Configuration** field, click **modify**:



2. On the Project Facets page, select **Java Persistence**, and click **Okay**.
3. Follow the steps to create your EJB project; on the JPA Facets page,
 - A. For your version of JPA, make the appropriate selection in the **Platform** field:
 - For JPA 1.0 or 2.0, select **RAD JPA Platform**.
 - For JPA 2.1, select either **Generic** or **EclipseLink**.
 - B. In the **Connection** field, select a connection, or click **Add Connection** to create a connection. Follow the steps to create a database connection of your choice.
 - C. Select **Override default schema from connection**, and select an alternative schema in the **Schema** field, if you do not want to use the default schema, and click **Finish**.
4. **Create a session bean in your EJB project:**
 - A. In the Java™ EE perspective, right-click your project, and select **New > Session Bean**. The Create EJB 3.x Session Bean wizard appears.
 - B. In the **Source folder** field, select the source folder for the new bean.
 - C. In the **Java package** field, type the package name for the new bean.
 - D. In the **Bean name** field, type the name that you want to assign to the enterprise bean. By convention, bean names begin with an uppercase letter. **Note:** You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes associated with enterprise beans.
 - E. Select Remote to add a remote interface and select Local to add a local interface, and click **Finish**.
5. **Create a JPA entity in your EJB project:**
 - A. Right-click your EJB project, and select **JPA > Generate entities...**
 - B. On the Database Connection page, ensure that the connection and schema are correct, and click **Next**.
 - C. On the Generate Entities from Tables page in the Source folder field, type a name for your source folder or browse to the path of the folder that contains the Java source file for your entity.
 - D. In the Java package field, type or browse to the Java package for your entities.
 - E. Select **Synchronize Classes in persistence.xml**, if you want to synchronize your entity class with the persistence.xml file.
 - F. In the **Tables** field, select the table or tables from which you want to create entities, and click **Finish**. The Java Editor opens with your JPA entity class.
6. **Create queries in your JPA entity class:** Open your JPA entity class in the Java Editor, and you can create queries to retrieve data from the database. For example, here are two simple queries in the entity class: @Entity

```
@NamedQueries({
    @NamedQuery(name = "findBySalaryLessThan", query = "SELECT e FROM Employee e WHERE e.salary < :salary"),
```

```
@NamedQuery(name = "findBySalaryGreaterThan", query = "SELECT e FROM Employee e WHERE e.salary > :salary")
})
```

7. **Use your JPA entity in your session bean** In your stateless session bean, you can use injection of EntityManager to run the queries that you created in your JPA entity. Open your session bean class in the Java Editor and create an EntityManager that connects to the JPA entity. Here is an example that calls the queries created in a previous step:

```
@Stateless
public class HumanResourcesBean implements HumanResources {

    @PersistenceContext
    private EntityManager emanager;

    public HumanResourcesBean() {
    }

    public List<Employee> findBySalaryLessThan(double salary) {
        Query query = emanager.createNamedQuery("findBySalaryLessThan");
        query.setParameter("salary", BigDecimal.valueOf(salary));

        @SuppressWarnings("unchecked")
        List<Employee> result = query.getResultList();
        return result;
    }

    public List<Employee> findBySalaryGreaterThan(double salary) {
        Query query = emanager.createNamedQuery("findBySalaryGreaterThan");
        query.setParameter("salary", BigDecimal.valueOf(salary));

        @SuppressWarnings("unchecked")
        List<Employee> result = query.getResultList();
        return result;
    }
}
```

Parent topic:[Creating enterprise beans using wizards](#)

Creating a stateful session bean using wizards

You can use the Create EJB 3.x Session Bean wizard to create a stateful session bean and add it to your project.

Before you begin

You must have a Java™ project, an EJB project, or a web project created in your workspace.

Procedure

1. In the Java EE perspective, right-click your project, and select **New > Session Bean**. The Create EJB 3.x Session Bean wizard opens.
2. In the **Source folder** field, select the source folder for the new bean.
3. In the **Java package** field, type the package name for the new bean.
4. In the **Bean name** field, type the name that you want to assign to the enterprise bean. By convention, bean names begin with an uppercase letter. **Note:** You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes associated with enterprise beans.
5. Select **Remote** to add a remote interface and select **Local** to add a local interface, and click **Finish**.
6. In the Java class editor, underneath the package declaration, you can see the `@Stateful` annotation. Your class also contains reference to **Local** and **Remote** interfaces, if you selected to create them:
`package com.ibm.test;`

```
import javax.ejb.Stateful;

/**
 * Session Bean implementation class TestBean
 */
@Stateful
public class TestBean implements TestBeanRemote, TestBeanLocal {

    /**
     * Default constructor.
     */
    public TestBean() {
        // TODO Auto-generated constructor stub
    }

}
```

7. Define the client views and interfaces. For EJB 3.0 or later beans, you can include a remote client interface, a local interface, or both. Here is an example of a simple Remote interface:
`package com.ibm.websphere.ejb3sample.counter;`

```
import javax.ejb.Remote;

@Remote
public interface RemoteCounter {
    public int increment();
    public int getTheValue();
}
```

Parent topic: [Creating enterprise beans using wizards](#)

Creating an EJB timer by using wizards

You can use the Create EJB Timer wizard to create Enterprise JavaBeans (EJB) Timers and add them to your project.

Before you begin

You must have a Java™ project, an EJB 3.1 project, an EJB 3.2 project, or a web project that is created in your workspace.

Procedure

1. In the Java EE perspective, right-click your project, and select **New > Other > EJB > EJB Timer**. The Create EJB Timer wizard opens.
2. In the **Source folder** field, select the source folder for the new bean.
3. In the **Java package** field, type the package name for the new bean.
4. In the **Class name** field, type the name that you want to assign to the enterprise bean. By convention, bean names begin with an uppercase letter.**Note:** You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes that are associated with enterprise beans.
5. In the **Schedule** field, modify the preinstalled calendar-based timer expressions for this timer.
6. Optional: Select **Non-persistent**.When you make the selection, the persistent element of the Schedule annotation is set to `false`.
7. Click **Finish**.

Results

In the Java class editor, in the `scheduleTimeout` method, you can see the `@Schedule` annotation with the calendar-based timer expressions typed in the wizard. If the Non-persistent option was checked, the persistent element is set to `false`.

```
package com.ibm.test;

import javax.ejb.Schedule;
import javax.ejb.Stateless;
import javax.ejb.Timer;

@Stateless
public class MyTimer {

    /**
     * Default constructor.
     */
    public MyTimer() {
        // TODO Auto-generated constructor stub
    }

    @SuppressWarnings("unused")
    @Schedule(second="*/10", minute="*", hour="8-23", dayOfWeek="Mon-Fri",
        dayOfMonth="*", month="*", year="*", info="MyTimer", persistent=false)
    private void scheduledTimeout(final Timer t) {
        System.out.println("@Schedule called at: " + new java.util.Date());
    }
}
```


Parent topic: [Creating enterprise beans using wizards](#)

Creating a message-driven bean using wizards

You can use the Create EJB 3.1 Message-Driven Bean wizard to create a message-driven bean and add it to your project.

Before you begin

You must have a Java™ project, an EJB project, or a web project created in your workspace.

About this task

The main difference between a message-driven bean and a session bean is that a message-driven bean has no local or remote interface. Instead, it has only a bean class.

Procedure

1. In the Java EE perspective, right-click your project, and select **New > Message-Driven Bean**. The Create EJB 3.1 Message-Driven Bean wizard opens.
2. In the **Source folder** field, select the source folder for the new bean.
3. In the **Package** field, type the package name for the new bean.
4. In the **Name** field, type the name that you want to assign to the message-driven bean. By convention, bean names begin with an uppercase letter.
5. In the **Destination name** field, type the name that you want to assign to the destination.
6. Select **JMS** to use the Java messaging service, or clear **JMS** to use another messaging service.
7. In the **Destination type** field, select **Queue** or **Topic** destination type, and click **Next**.
8. In the Message-Driven bean-specific information page, in the **Transaction type** field, select **Container** for container-managed transactions or **Bean** for bean-managed transactions.
9. Click **Finish**. The Java editor contains the default code for your message-driven bean class:`package com.ibm.test;`

```
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

/**
 * Message-Driven Bean implementation class for: TestMdb
 */
@MessageDriven(
    activationConfig = { @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Queue"
    ) })
public class TestMdb implements MessageListener {

    /**
     * Default constructor.
     */
    public TestMdb() {
        // TODO Auto-generated constructor stub
    }
}
```

```

/**
 * @see MessageListener#onMessage(Message)
 */
public void onMessage(Message message) {
    // TODO Auto-generated method stub

}
}

```

In this example of default message-driven bean code, the following points are important to note:

- In EJB 3.1, the `@MessageDriven` annotation specifies a set of activation configuration parameters. These parameters are unique to the particular type of JCA 1.5 adapter that is used to drive the Message-driven bean. Some adapters have configuration parameters that let you specify the destination queue of the Message-driven bean. In the case where the adapter does not support this, the destination name must be specified using a `<message-destination>` entry in the XML binding file.
- The bean class has to implement the `MessageListener` interface, which defines only one method, `onMessage`. When a message arrives in the queue monitored by this MDB, the container calls the `onMessage` method of the bean class and passes the incoming message in as the parameter.
- The `ActivationConfigProperty` of the `@MessageDriven` annotation provides messaging system–specific configuration information.

Parent topic: [Creating enterprise beans using wizards](#)

Packaging EJB content in web application archive (WAR) modules

EJB functionality that is supported for beans packaged inside EJB JAR modules is also supported for beans packaged inside WAR modules. A bean packaged inside a WAR module is capable of the same behavior as a bean packaged inside an EJB JAR module.

Before you begin

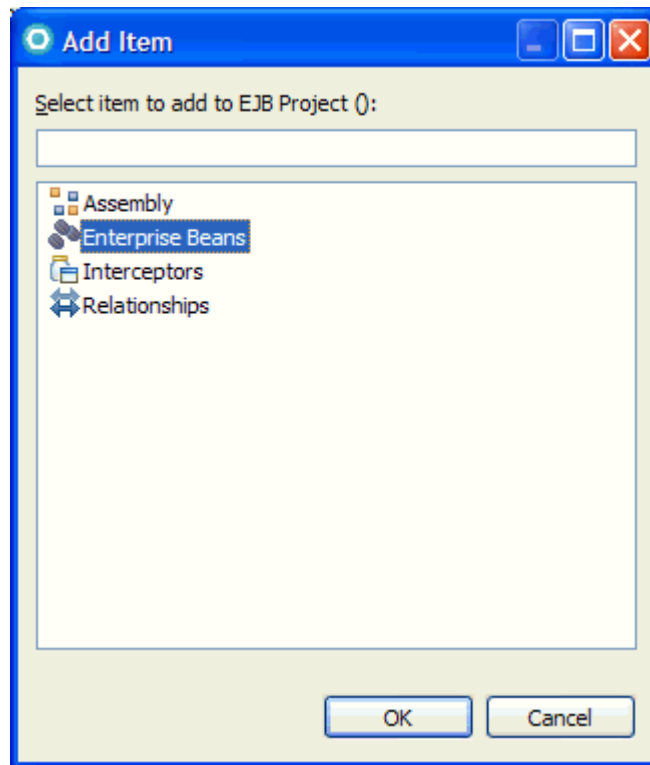
You must have a web project created in your workspace.

About this task

You can now place EJB classes directly in the .WAR file, using the same packaging guidelines that apply to web application classes. You can place EJB classes under the WEB-INF/classes directory or in a JAR file within the WEB-INF/lib directory. The EJB deployment descriptor is also optional. If you need it, you can package the EJB deployment descriptor as a WEB-INF/ejb-jar.xml file. The rules for packaging EJB content in a WAR module are different from the rules for packaging EJB content in a JAR module. For more information about packaging EJB content in WAR files, see [EJB content in WAR modules](#)

Procedure

1. In the Java™ EE perspective, right-click your Web project and select **New > Other > Session Bean (EJB 3.x)**, or **Message-Driven Bean (EJB 3.x)**, and click **Next**.
2. On the Create EJB 3.x Session Bean page or the Create Message-Driven Bean 3.x page, complete the following steps.
 - A. In the Java EE perspective, right-click your project, and select **New > Session Bean** or **New > Message-Driven Bean**. The Create EJB 3.x Session Bean or Create EJB 3.x Message-Driven Bean wizard appears.
 - B. In the **Source folder** field, select the source folder for the new bean.
 - C. In the **Java package** field, type the package name for the new bean.
 - D. In the **Bean name** field, type the name that you want to assign to the enterprise bean. By convention, bean names begin with an uppercase letter. **Note:** You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes associated with enterprise beans.
 - E. Select Remote to add a remote interface and select Local to add a local interface, and click **Finish**.
3. To create a deployment descriptor for your EJB, right-click your web project and select **Java EE > Generate EJB Deployment Descriptor Stub**. An `ejb-jar.xml` file appears in the `webContent/WEB-INF` folder.
4. Adding a session bean or message-driven bean using the deployment descriptor:
 - A. Right-click the `ejb-jar.xml` file, and select **Open With > EJB Deployment Descriptor Editor**.
 - B. Click **Add**. In the Add Item page, select **Enterprise Beans**:



C. In the title of the page, click **1 error detected**.

D. For **Session Bean**, click **Add**, then click **OK**.

E. Navigate to your web project, and expand `EJBs/Session Beans`, and your new session bean appears.

Parent topic: [Creating enterprise beans using wizards](#)

Deleting EJB 3.x beans

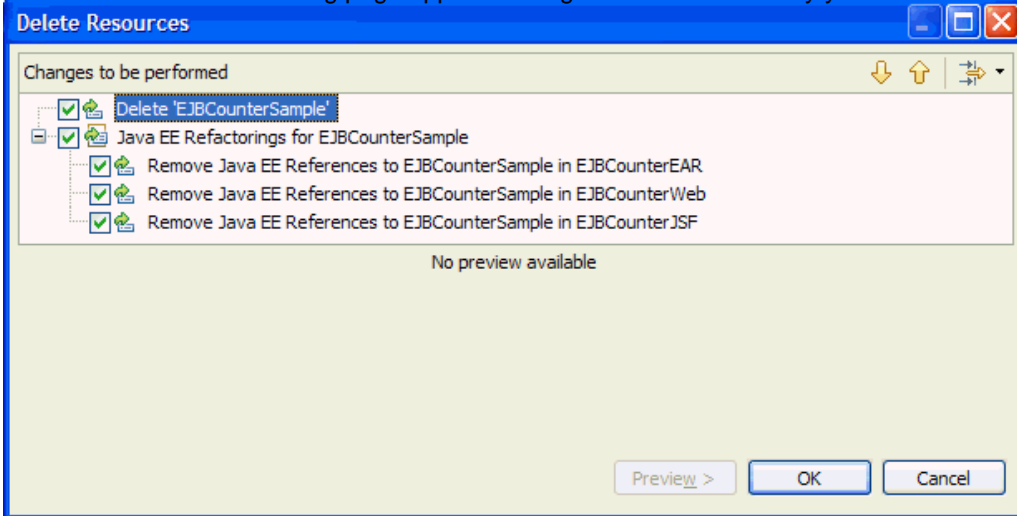
Once you have created an Enterprise Java™ bean, you might want to delete it. The instructions in this document describe the basic steps that you can take to delete the bean.

About this task

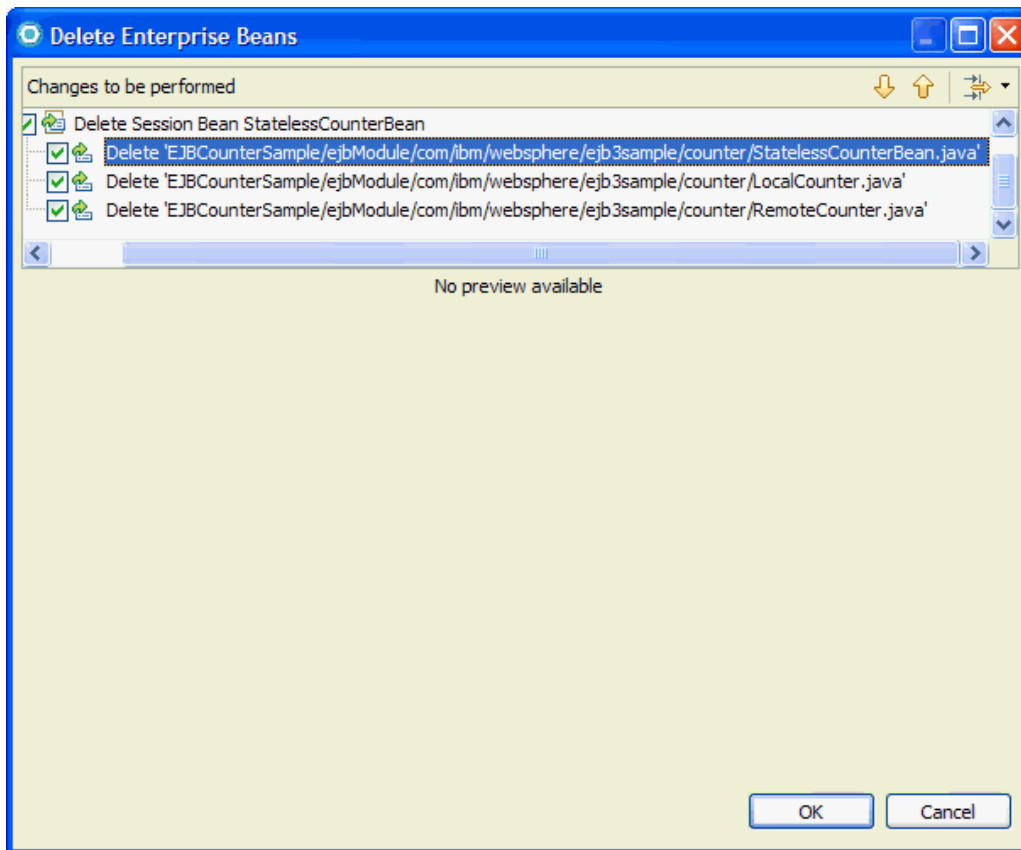
When you create an EJB project, code is generated in files as part of the process of creating the project. After you delete a module or a bean in a Java EE project, the affected files are also cleaned up. To delete a web module, EJB module, Application Client module, Connector module, or EJB client, or utility module:

Procedure

1. Right-click the EJB project that you want to delete, and select **Delete**. Alternatively, you can select the module in the project view, and press `Delete`. A dialog box appears, confirming the delete.
2. Select **Preview**. A refactoring page appears listing the items affected by your delete action.



3. To delete an individual bean from your EJB project, locate the bean under your project folder, right-click the bean that you want to delete, and select **Delete**. Alternatively, you can select the bean in the project view, and press `Delete`. A dialog box appears, confirming the delete.



Parent topic: [Creating enterprise beans using wizards](#)

Parent topic: [Editing EJB 3.x applications](#)

Creating enterprise beans using annotations

After you have created your Java™ or EJB project, you can create session beans, message-driven beans, and JPA entities to add to your project.


Before you begin

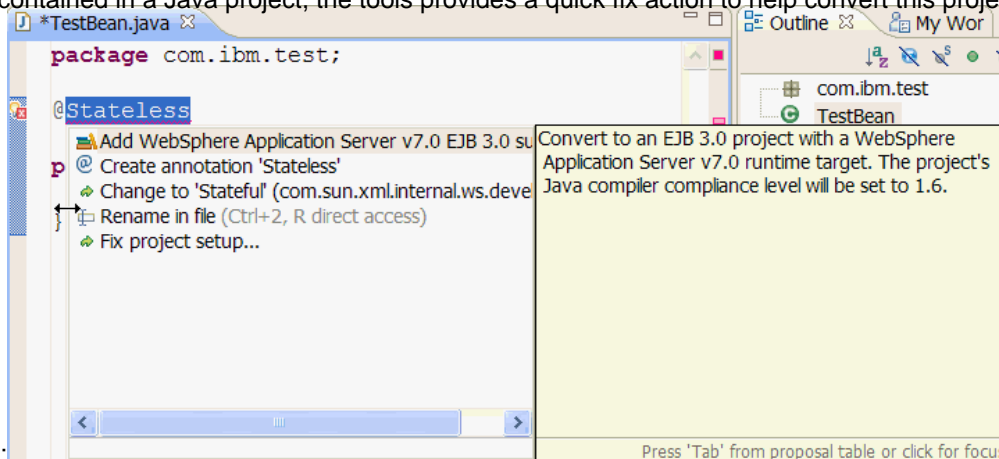
You must have a Java project, an EJB project, or a web project created in your workspace.

About this task

In the earlier versions of Enterprise Java bean specifications, two interfaces, home and remote, were defined for accessing the enterprise bean. They can be remote or local depending on the way the client accesses the bean. In EJB 3.1 specification, the home or remote interface is not required: only one interface is defined, the business interface. The business interface is a simple POJI (Plain Old Java Interface) and the type of the business interface (remote or local) is specified using annotations. All annotations required for writing EJB are defined in the javax.ejb package. Using these annotations, you can create session beans, message-driven beans, or entity beans.

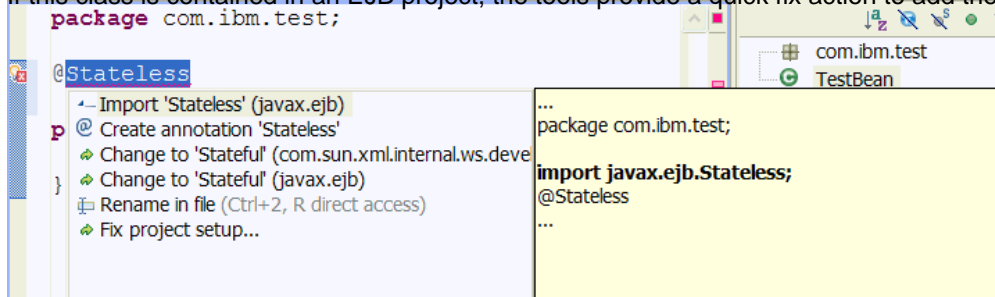
Procedure

1. The first step in creating an enterprise bean is to create a simple Java class. Right-click your project, and select **New** > **Class**.
2. Add a component-defining annotation, which indicates to the tools that this Java class should be treated as an EJB. Component-defining annotations for EJBs include:
 - **@Stateless**: Component-defining annotation for a stateless session bean.
 - **@Stateful**: Component-defining annotation for a stateful session bean.
 - **@MessageDriven**: Component-defining annotation for a message driven bean.
3. Right-click the  Quick-fix icon, and select the appropriate action for you project:
 - If this class is contained in a Java project, the tools provides a quick fix action to help convert this project to an EJB



Select **Add WebSphere Application Server v7.0 EJB 3.0 support**, and your Java project is converted into an EJB 3.0 project, and quick fix and content assist are available for all EJB 3.0 annotations while in the Java Editor.

- If this class is contained in an EJB project, the tools provide a quick fix action to add the required import statement:



Select **Import 'Stateless' (javax.ejb)**, and the import statement `import javax.ejb.Stateless;` is added to your class.

- **Creating a stateless session bean using annotations**

You can use Java EE annotations to create a stateless session bean and add it to your project.

- **Creating a stateful session bean using annotations**

You can use Java EE annotations to create a stateful session bean and add it to your project.

- **Creating a message-driven bean using annotations**

You can use Java EE annotations to create a message-driven bean and add it to your project.

- **Common annotations**

The `javax.annotation` package defines common annotations.

- **EJB annotations**

The `javax.ejb` package contains the enterprise JavaBeans classes and interfaces that define the contracts between the enterprise bean and its clients and between the enterprise bean and the EJB container.

Parent topic: [Creating enterprise beans](#)


Creating a stateless session bean using annotations

You can use Java™ EE annotations to create a stateless session bean and add it to your project.

Before you begin

You must have a Java project, an EJB project, or a web project created in your workspace.

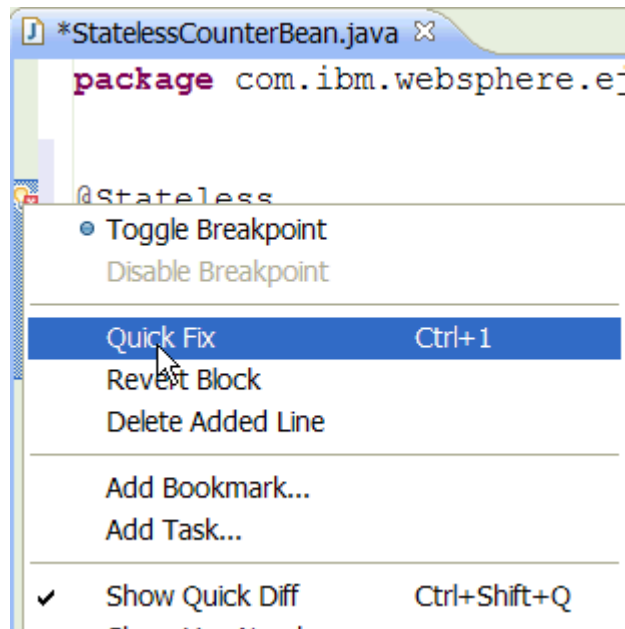
Procedure

1. In the Enterprise Explorer view, click **File > New > Class**. The Create a New Java Class wizard opens.
2. In the **Source folder** field, select the source folder for the new bean.
3. In the **Java package** field, type the package name for the new bean.
4. In the **Bean name** field, type the name that you want to assign to the enterprise bean. By convention, bean names begin with an uppercase letter, and click **Finish**. **Note:** You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes associated with enterprise beans.
5. In the Java class editor, underneath the package declaration, type `@Stateless`. You can see an error / quick-fix icon  next to the `@Stateless` line. **Tip:** You can simply type `@Sta` and then press CTRL+Spacebar to see the options in context assistance:

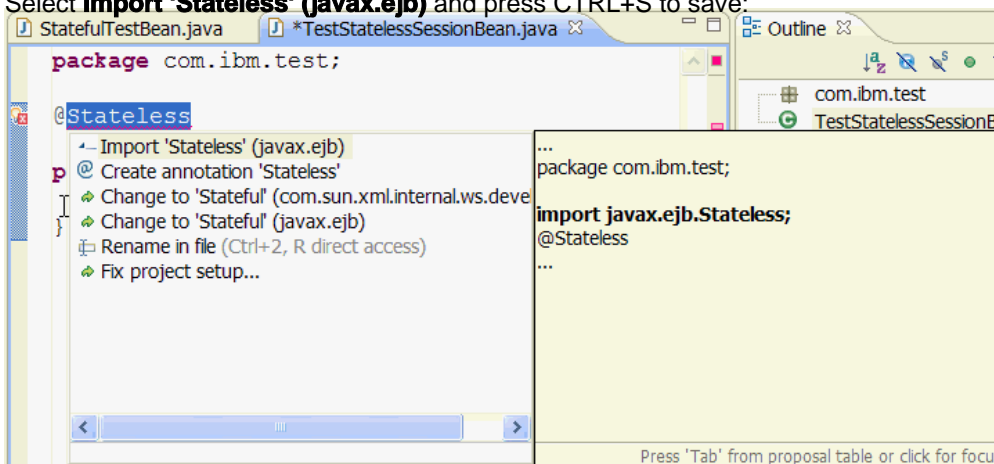


Select **@Stateless(EJB) - javax.ejb** to create a Stateless session bean.

6. When you press CTRL+S to save, you can see a quick-fix icon  next to the `@Stateless` line.
7. Right-click the quick-fix icon and select **Quick Fix**:



8. Select **Import 'Stateless' (javax.ejb)** and press CTRL+S to save:



9. Select **Import 'Stateless' (javax.ejb)**, press CTRL+S to save.

10. In the Enterprise Explorer view, expand your **<EJB project_name> > ejbModule**, and your new session bean Java class is listed under its package name.

11. Define the client views and interfaces. For EJB 3.0 or later beans, you can include a remote client interface, a local interface, or both. Here is an example of a remote interface:

```
package com.ibm.test;


import javax.ejb.Remote;

@Remote
public interface TestBeanRemote {




}
```

- **Remote client interface:** To create a remote client interface:

- Right-click your EJB project, select click **New > Interface**. In the Create a New Java Interface wizard, type the package name of your session bean in the **Package** field. Type a name for your interface in the **Name** field and click **Next**.
- In the Interface editor, type `@Remote` to your new remote interface, after the package declaration. When you press CTRL+S to save, you can see a quick-fix icon with the `@Remote` line. Right-click the quick-fix icon and select **Quick Fix**, select **Import 'Remote' (javax.ejb)** and press CTRL+S to save.
- To add a Remote home interface, add the annotation `@RemoteHome` to your session bean class. When you press CTRL+S to save, you can see a quick-fix icon with the `@RemoteHome` line. Right-click the quick-fix icon and

select **Quick Fix**, select **Import 'RemoteHome' (javax.ejb)** and press CTRL+S to save. When you press CTRL+S to save, you can see a quick-fix icon  with the @RemoteHome line. Right-click the quick-fix icon and select **Quick Fix**, select **Add missing attributes**. Provide the values for the name-value pair: (*value=null*), and press CTRL+S to save.

- **Local client interface**: To create a local client interface:

- A. Right-click on your EJB project, select click **New > Interface**. In the Create a New Java Interface wizard, type the package name of your session bean in the **Package** field. Type a name for your interface in the **Name** field and click **Next**.
- B. In the Interface editor, type @Local to your new local interface, after the package declaration. When you press CTRL+S to save, you can see a quick-fix icon  next to the @Local line. Right-click the quick-fix icon and select **Quick Fix**, select **Import 'Local' (javax.ejb)** and press CTRL+S to save.
- C. To add a Local home interface, add the annotation @LocalHome to your session bean class. When you press CTRL+S to save, you can see a quick-fix icon  with the @LocalHome line. Right-click the quick-fix icon and select **Quick Fix**, select **Import 'LocalHome' (javax.ejb)** and press CTRL+S to save. When you press CTRL+S to save, you can see a quick-fix icon  with the @LocalHome line. Right-click the quick-fix icon and select **Quick Fix**, select **Add missing attributes**. Provide the values for the name-value pair: (*value=null*), and press CTRL+S to save.

Parent topic: [Creating enterprise beans using annotations](#)


Creating a stateful session bean using annotations

You can use Java™ EE annotations to create a stateful session bean and add it to your project.

Before you begin

You must have a Java project, an EJB project, or a web project created in your workspace.

Procedure


1. In the Java EE perspective, click **File > New > Class**. The Create a New Java Class wizard opens.
2. In the **Source folder** field, select the source folder for the new bean.
3. In the **Java package** field, type the package name for the new bean.
4. In the **Bean name** field, type the name that you want to assign to the enterprise bean. By convention, bean names begin with an uppercase letter. **Note:** You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes associated with enterprise beans.
5. In the Java class editor, underneath the package declaration, type `@stateful`. You can see an error / quick-fix icon  next to the `@stateful` line. **Tip:** You can simply type `@Sta` and then press CTRL+Spacebar to see the options in context assistance:

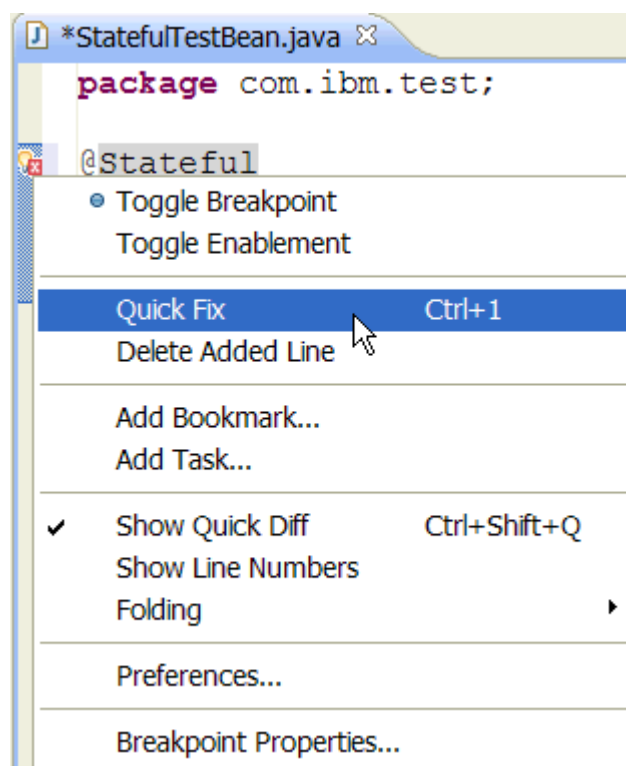


Select **@Stateful(EJB) - javax.ejb** to create a stateful session bean.

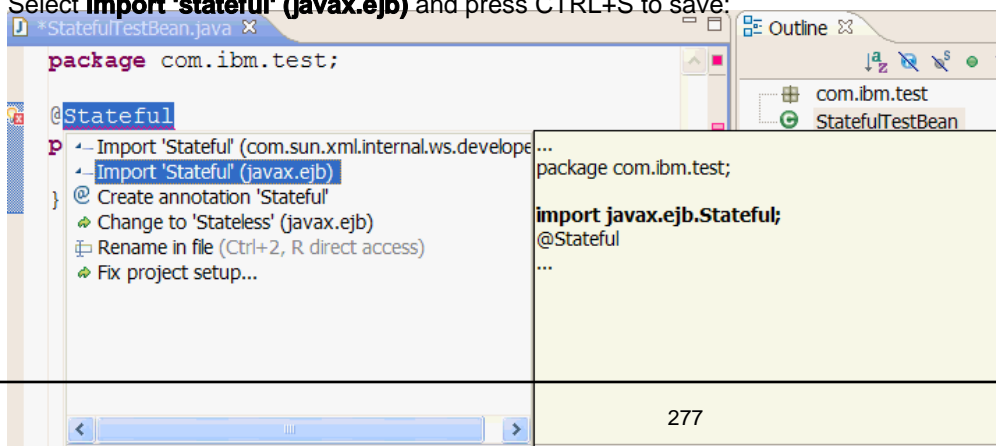
6. Press CTRL+Spacebar to see the options in context assistance:



7. Select **@Stateful(EJB) - javax.ejb** to create a stateful session bean.
8. When you press CTRL+S to save, you can see a quick-fix icon  next to the @Stateful line.
9. Right-click the quick-fix icon and select **Quick Fix**:



10. Select **Import 'stateful' (javax.ejb)** and press CTRL+S to save:



11. In the Enterprise Explorer view, expand your **<EJB project_name>** > **ejbModule**, and your new session bean Java class is listed under its package name.
12. Define the client views and interfaces. For EJB 3.0 or later beans, you can include a remote client interface, a local interface, or both. Here is an example of a basic remote interface:

```
package com.ibm.test;

import javax.ejb.Remote;

@Remote
public interface TestBeanRemote {

}
```

- **Remote client interface:** To create a remote client interface:
 - A. Right-click your EJB project, select click **New > Interface**. In the Create a New Java Interface wizard, type the package name of your session bean in the **Package** field. Type a name for your interface in the **Name** field and click **Next**.
 - B. In the Interface editor, type `@Remote` to your new remote interface, after the package declaration. When you press CTRL+S to save, you can see a quick-fix icon with the `@Remote` line. Right-click the quick-fix icon and select **Quick Fix**, select **Import 'Remote' (javax.ejb)** and press CTRL+S to save.
 - C. To add a Remote home interface, add the annotation `@RemoteHome` to your session bean class. When you press CTRL+S to save, you can see a quick-fix icon with the `@RemoteHome` line. Right-click the quick-fix icon and select **Quick Fix**, select **Import 'RemoteHome' (javax.ejb)** and press CTRL+S to save. When you press CTRL+S to save, you can see a quick-fix icon with the `@RemoteHome` line. Right-click the quick-fix icon and select **Quick Fix**, select **Add missing attributes**. Provide the values for the name-value pair: `(value=null)`, and press CTRL+S to save.
- **Local client interface:** To create a local client interface:
 - A. Right-click on your EJB project, select click **New > Interface**. In the Create a New Java Interface wizard, type the package name of your session bean in the **Package** field. Type a name for your interface in the **Name** field and click **Next**.
 - B. In the Interface editor, type `@Local` to your new local interface, after the package declaration. When you press CTRL+S to save, you can see a quick-fix icon with the `@Local` line. Right-click the quick-fix icon and select **Quick Fix**, select **Import 'Local' (javax.ejb)** and press CTRL+S to save.
 - C. To add a Local home interface, add the annotation `@LocalHome` to your session bean class. When you press CTRL+S to save, you can see a quick-fix icon with the `@LocalHome` line. Right-click the quick-fix icon and select **Quick Fix**, select **Import 'LocalHome' (javax.ejb)** and press CTRL+S to save. When you press CTRL+S to save, you can see a quick-fix icon with the `@LocalHome` line. Right-click the quick-fix icon and select **Quick Fix**, select **Add missing attributes**. Provide the values for the name-value pair: `(value=null)`, and press CTRL+S to save.

Parent topic: [Creating enterprise beans using annotations](#)

Creating a message-driven bean using annotations

You can use Java™ EE annotations to create a message-driven bean and add it to your project.


Before you begin

You must have a Java project, an EJB project, or a web project created in your workspace.

About this task

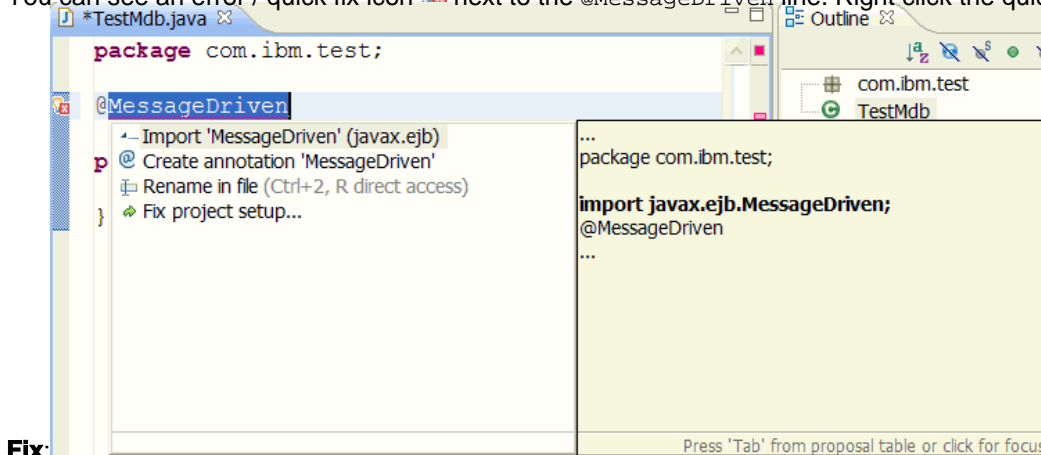
The main difference between a message-driven bean and a session bean is that a message-driven bean has no local or remote interface. Instead, it has only a bean class.

Procedure

1. In the Java EE perspective, click **File > New > Class**.
2. In the **Source folder** field, select the source folder for the new bean.
3. In the **Default package** field, type the package name for the new bean, and click **Finish**.
4. In the Java class editor, underneath the package declaration, type `@MessageDriven`. You can see an error / quick fix icon  next to the `@MessageDriven` line. **Tip:** You can simply type `@Mess` and then press **CTRL+Spacebar** to see the



5. You can see an error / quick fix icon  next to the `@MessageDriven` line. Right click the quick fix icon and select **Quick**



6. Select **@MessageDriven(EJB)** and the tools automatically add the dependency **import javax.ejb.MessageDriven;**
7. In the Enterprise Explorer view, expand your **<Java project_name> > ejbModule**, and your new message-driven bean Java class is listed under its package name.
8. You can use the `@MessageDriven` annotation to specify properties for the bean, including
 - Destination type

- A durable subscription
- A message selector
- An acknowledgment mode

Parent topic: [Creating enterprise beans using annotations](#)

Common annotations

The `javax.annotation` package defines common annotations.

For information on the available annotation tags from this package for either Java™ EE 6 or Java EE 7, see the [Oracle Java EE API Documentation](#).

Parent topic: [Creating enterprise beans using annotations](#)

EJB annotations

The `javax.ejb` package contains the enterprise JavaBeans classes and interfaces that define the contracts between the enterprise bean and its clients and between the enterprise bean and the EJB container.

For information on the available EJB 3.1 annotation tags in Java™ EE 6 and Java EE 7, see the `javax.ejb` and `javax.interceptor` packages in the [Oracle Java EE API Documentation](#).

Parent topic: [Creating enterprise beans using annotations](#)

Importing class files to Web projects

About this task

You can use the Import wizard to add .class files to a Web project. With the wizard, you can import the class files from the directory system or from a JAR file or archive file. After you choose the class files that you want to import, the wizard creates an **imported_classes** folder, imports the specified class files to that folder, and adds the folder to the default classpath. You can navigate the **imported_classes** folder in the Enterprise Explorer view.

Windows You can drag and drop class files from the Windows Explorer or desktop to the **imported_classes** folder in the Project Navigator view.

To import class files to a Web project using the wizard:

Procedure

1. Right-click your Web project and select **Import > Import Class Files ...** from the pop-up menu. The Import wizard opens.
2. Specify whether you want to **Import from directory** or **Import from ZIP or JAR** and click **Next**.
3. Depending on whether you are importing from a directory or JAR file or archive file, in the **Source** field enter the full path of the directory, archive file, or JAR file that contains the .class files that you want to import. You can click the **Browse** button to locate the directory or file on your system.
4. Select the class files that you want to import.
5. Optional: If you have previously imported class files and do not want to be warned about overwriting resources with the same names, select **Overwrite existing resources without warning**.
6. Click **Finish**. The wizard imports the .class files to the **imported_classes** folder.

Parent topic: [Creating enterprise beans](#)

Editing EJB 3.x applications

You can edit your EJB 3.x applications in your source code by directly modifying your source code and the annotations in the Java™ editor or by using the Annotations view.

- **Using as-you-type validation**

The Java EE tools feature enhanced EJB 3.x as-you-type validation. While you modify the annotations for your EJB, as-you-type validation runs to provide you feedback on any problems that it detects. These problems become errors when you save them.

- **Content assist and EJB 3.x**

The content assist tool is a feature of the workbench. You can type code in your Java editor and the content assist tool recommends possible ways to complete your code.

- **Deleting EJB 3.x beans**

Once you have created an Enterprise Java bean, you might want to delete it. The instructions in this document describe the basic steps that you can take to delete the bean.

- **Refactoring EJB 3.x Java elements**

Using workspace refactoring operations, you can quickly and safely rename code artifacts globally across your application. The workspace recognizes EJB 3.x applications and manages changes to relevant annotations and deployment descriptors.

Parent topic: [Developing EJB 3.x applications](#)

Using as-you-type validation

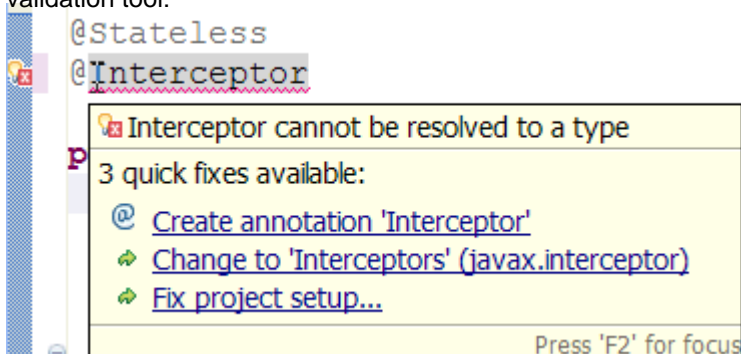
The Java™ EE tools feature enhanced EJB 3.x as-you-type validation. While you modify the annotations for your EJB, as-you-type validation runs to provide you feedback on any problems that it detects. These problems become errors when you save them.

Before you begin

In the Java EE perspective, open your Java class in the Java editor.

Procedure

1. Type your code. While you modify the EJB 3.x annotations in your Java class, as-you-type validation simultaneously runs and tracks any errors that you might have in your code. The validator flags any statement or value that has syntactic or usage problems by underlining it in red.
2. Select one of the quick fixes that the validator suggests. The following screen capture illustrates the as-you-type validation tool:



Parent topic: [Editing EJB 3.x applications](#)

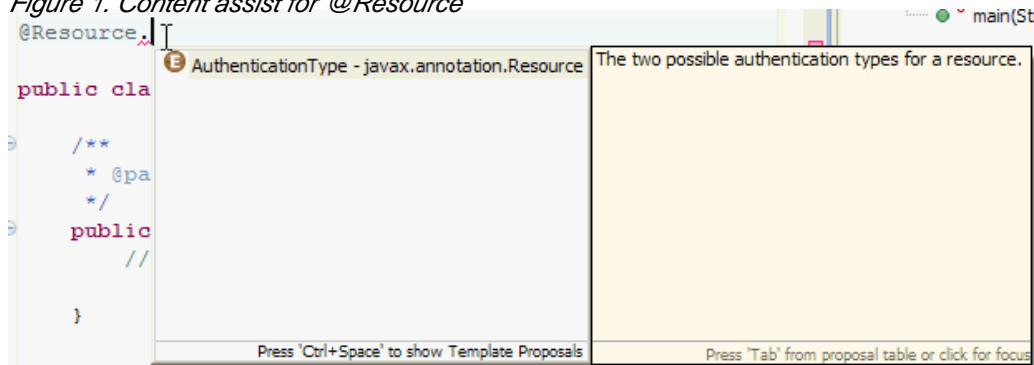
Content assist and EJB 3.x

The content assist tool is a feature of the workbench. You can type code in your Java™ editor and the content assist tool recommends possible ways to complete your code.

With the introduction of Java EE 5, the content assist tool in the application development workbench has been enhanced. Content assist now recognizes EJB 3.x structures. It provides content assist content when the project has the EJB 3.x facet enabled, and provides component defining content assist when the project does not have any Java EE facets enabled.

Content assist template support also exists for some of the EJB 3.x annotations, such as @Resource, to provide an annotation with initial attribute values.

Figure 1. Content assist for @Resource



Parent topic: [Editing EJB 3.x applications](#)

Refactoring EJB 3.x Java elements

Using workspace refactoring operations, you can quickly and safely rename code artifacts globally across your application. The workspace recognizes EJB 3.x applications and manages changes to relevant annotations and deployment descriptors.

Procedure

1. In the **Enterprise Explorer** view, the **Outline** view, or the Java™ editor, right-click the field or method you want to rename and select **Refactor**. From the submenu that follows, select **Rename**. The **Rename Field** or **Rename Method** panel opens.
2. Enter the new name for the field or method and click **Next**.
3. Review each change to be performed. If you do not want to make a particular change, clear the box next to the change. Click **Finish**.

What to do next

The operation makes changes to the Java artifacts that reference the field or method that you rename. If the field or method is annotated with a dependency injection annotation (`@EJB`, `@Inject`, or `@Resource`), the refactoring operation also updates the XML artifacts that make reference to the field or method. Additionally, by selecting **Java EE Tools > Promote Method** from the context menu of a method, you can move the method to the business interface for the EJB. If a bean class specifies its business interfaces using annotations or the EJB deployment descriptor, you can still specify the business interface as the target of the operation. A **Refactor > Move** operation moves a field or method from one package to another.

Parent topic: [Editing EJB 3.x applications](#)

EJB Security

You can provide security for your EJB application using annotations or using deployment descriptors.

Before Java™ EE 5, if you wanted to use authorization for a given application, you needed to specify authorization information in the application deployment descriptors `ejb-jar.xml` or `web.xml`. You can set up security in your application directly using annotations.

Common security annotations

JSR 250 defines a number of common security annotations. Five security annotations are defined:

- **`javax.annotation.security.PermitAll`:**

- Can be used at type or method level.
- Indicates that the given method or all business methods of the given EJB are accessible by everyone.

- **`javax.annotation.security.DenyAll`:**

- Can be used at method level.
- Indicates that the given method in the EJB cannot be accessed by anyone.

- **`javax.annotation.security.RolesAllowed`:**

- Can be used at type or method level.
- Indicates that the given method or all business methods in the EJB can be accessed by users associated with the list of roles.

- **`javax.annotation.security.DeclareRoles`:**

- Can be used at type level.
- Defines roles for security checking. To be used by `EJBContext.isCallerInRole`, `HttpServletRequest.isUserInRole`, and `WebServiceContext.isUserInRole`.

- **`javax.annotation.security.RunAs`:**

- Can be used at type level.
- Specifies the run-as role for the given components.

Example: `@Stateless`

```
@RolesAllowed("team")

public class TestEJB implements Test {

    @PermitAll

    public String hello(String msg) {

        return "Hello, " + msg;

    }

    public String goodbye(String msg) {

        return "Goodbye, " + msg;

    }

}
```

In this example, the `hello()` method is accessible by everyone, and the `goodbye()` method is accessible by users of role `team`.

- Securing EJBs

You can provide security for your EJB application using annotations or using deployment descriptors.

Parent topic: [Enterprise application security](#)

Testing EJB 3.x applications

After you have created an EJB 3.x applications, you can create a Servlet or JSF to test the EJB 3.x applications.

About this task

To test an EJB application with a servlet, you must first inject an EJB reference using the `@EJB` injection annotation from EJB 3.x. Once you have injected the EJB, you can call the methods that are available from the Remote or Local interface.

Procedure

- Testing EJB 3.x applications with a servlet

1. Select **File > New > Other** and select **Servlet**.
2. Type the package and class name for your servlet.
3. After the servlet has been created and the Java™ editor opened on the servlet class, insert the `@EJB` annotation tag along with the reference to the local or remote interface class as a new field in the servlet.
4. You can now invoke any of the methods in the local or remote interface from within the `doPost()` or `doGet()` methods of your servlet. The following snippet is taken from the EJB 3.x Counter sample. You will see the `statelessCounter` field is declared with the type being `LocalCounter` which is our local interface for the EJB. The `@EJB` annotation in front of it injects an instance of it into our servlet.

```
// Use injection to get the ejb
@EJB private LocalCounter statelessCounter;
```

- Testing EJB 3.x applications with a JSF file This type of testing currently requires some manual configuration steps. For an example of the code involved, import the EJB 3.x Counter sample from the information center and locate the `EJBCounter.jsf` file in the `WebContent` folder as well as the page code Java classes in the Java sources in the `EJBCounter.jsf` project.

- Testing EJB 3.1 applications

After you have created a EJB 3.1 applications, you can test your application using an application server.

Parent topic: [Developing EJB 3.x applications](#)

Testing EJB 3.1 applications

After you have created a EJB 3.1 applications, you can test your application using an application server.

Before you begin

For information about testing Java™ EE application on a server, see [Testing your application](#). **Note:** You can test enterprise beans at 2.1 (or earlier) specification-level in the Universal Test Client running on a WebSphere® Application Server v6.1 or v7.0. However, you can only test EJB 3.1 beans in the Universal Test Client running on a WebSphere Application Server v7.0. The Universal Test Client does not support testing EJB 3.1 beans on a WebSphere Application Server v6.1 with Feature Pack for EJB 3.1.

Parent topic: [Testing EJB 3.x applications](#)

Developing Java Persistence API (JPA) applications

You can develop applications with the Java™ Persistence API (JPA), which is a simplification of the persistence programming model. These topics provide information on creating and configuring JPA projects and entity beans, and other features of the JPA tools.

- [Learn about JPA applications](#)

The Java Persistence API (JPA) defines the management of persistence and object/relational mapping within Java Enterprise Edition (Java EE) and Java Standard Edition (Java SE) environments.

- [JPA architecture](#)

The Java Persistence API represents a simplification of the persistence programming model.

- [Creating JPA projects](#)

You can use the JPA Project wizard to create and configure a JPA project.

- [Adding JPA support to a project](#)

You can add Java Persistence API (JPA) support to a faceted project by adding the Java Persistence project facet.

- [Converting a Java project to a JPA project](#)

You can convert a plain Java project to a JPA-enabled project.

- [Changing a JPA 1.0 project to JPA 2.0](#)

You can change a JPA 1.0 faceted project to JPA 2.0

- [Creating JPA entity beans](#)

You can create JPA entity beans from a wizard, by adding persistence to a POJO, or from existing database tables.

- [Configuring JPA entity beans](#)

- [Adding a primary key to the JPA entity](#)

Every entity that is mapped to a relational database must have a mapping to a primary key in the table.

- [Synchronizing persistent classes in the persistence.xml file](#)

After you create persistent classes, you can automatically add them to list of classes in their persistence unit in the persistence.xml file.

- [Working with persistence units \(persistence.xml\)](#)

You can edit the persistence.xml file with the Persistence XML Editor.

- [Adding or overriding entity mappings in the orm.xml file](#)

You can use the Object Relational Mapping XML Editor to define object-relational mappings for JPA entity beans in the orm.xml file.

Parent topic: [Developing enterprise applications](#)

Learn about JPA applications

The Java™ Persistence API (JPA) defines the management of persistence and object/relational mapping within Java Enterprise Edition (Java EE) and Java Standard Edition (Java SE) environments.

The Java Persistence API (JPA) represents a simplification of the persistence programming model. JPA manages persistence and object/relational mapping within the Java EE specification for Enterprise Java Beans 3.0. The JPA specification defines the object/relational mapping within its own guidelines instead of relying on vendor-specific mapping implementations. These features make applications that use JPA easier to implement and manage.

JPA combines the best features from previous persistence mechanisms such as Java Database Connectivity (JDBC) APIs, Object Relational Mapping (ORM) frameworks, and Java Data Objects (JDO). Creating entities under JPA is as simple as creating serializable classes. JPA supports the large data sets, data consistency, concurrent use, and query capabilities of JDBC. Like object-relational software and object databases, JPA allows the use of advanced object-oriented concepts such as inheritance. JPA avoids vendor lock-in because it does not rely on a strict specification like JDO and EJB 2.x entities.

The JPA implementation does not mandate that you migrate existing applications. Existing EJB 2.x Container Manager Persistence applications continue to run without changes. JPA might not be ideal for every application, however, for many applications it provides a better alternative to other persistence implementations.

With the JPA tools in the product, you can use wizards to create and automatically initialize mappings. You can create new database tables from existing entity classes (top-down mapping) or new entity beans from existing database tables (bottom-up mapping). You can also use the tools to create mappings between existing database tables and entity beans (meet-in-the-middle mapping), where names or other attributes differ. For flexibility in designing your data access application, you can choose from a range of mapping types. You can create mappings from several types of Java class, and you can specify entity inheritance with several options for database design.

JPA is covered under the JSR 220 EJB 3.0 specification: [JSR 220: Enterprise JavaBeans 3.0](#)

Overview

You can read the following topics before creating a JPA applications. They provide planning and technology overview information that might be useful if you are new to JPA applications or developing JPA applications in this development environment.

- [JPA architecture](#)

- The Java Persistence API represents a simplification of the persistence programming model.

Getting started

If you are already familiar with JPA applications technology the following topics will help you set up your workspace for JPA applications development, and guide you through the development process.

Resources for learning available on the Web

See the following links for more information:

[IBM® Redbooks®: WebSphere® Application Server Version 6.1 Feature Pack for EJB 3.0](#)

Note: [Latest developerWorks® articles and tutorials about JPA in Rational® Application Developer](#)

Note: [Latest developerWorks articles and tutorials about JPA](#)

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

JPA architecture

The Java™ Persistence API represents a simplification of the persistence programming model.

Data persistence, the ability to maintain data between application sessions in some form of nonvolatile storage (such as a relational database), is crucial to enterprise applications. Applications that are developed for this environment must either manage data persistence themselves or make use of third-party solutions to handle database updates and retrievals. JPA provides a mechanism for managing data persistence and object-relational mapping and functions for the EJB 3.0 specifications.

JPA is based on the Java programming model that applies to Java EE environments, but JPA can also function within the Java SE environment. The JPA specification defines the object-relational mapping internally, rather than relying on vendor-specific mapping implementations, and uses either annotations or XML to map objects into database tables.

JPA is designed to operate both inside and outside of a Java Enterprise Edition (Java EE) container. When you run JPA inside a container, applications can use the container to manage the persistence. If there is no container to manage JPA, the application must handle the persistence management itself. Applications that are designed for container managed persistence cannot be used outside a container, while applications that manage their own persistence can function either in a container environment or a Java SE environment.

JPA also provides a query language - JPQL - that you can use to retrieve objects without writing SQL queries specific to the database you are working with.

Java EE containers that support JPA must supply a persistence provider. A JPA persistence provider uses the following elements to persist data in an EJB 3.0 environment:

- **Entity objects:** An entity is a simple Java class that represents a row in a database table. Entities can be concrete classes or abstract classes. They maintain states by using properties or fields.
- **EntityManager:** An EntityManager object maintains the active collection of entity objects that are being used by the application. The EntityManager object handles the database interaction and metadata for object-relational mappings. An instance of an EntityManager object represents a persistence context. An application in a container can obtain the EntityManager either through injection into the application or by looking it up in the Java component namespace. If the application manages its persistence, the EntityManager is obtained from the EntityManagerFactory. The application server containers typically supply this function, but the EntityManagerFactory is required if you are using JPA application-managed persistence. **Note:** Injection of the EntityManager is only supported for the following artifacts:
 - EJB 3.0 session beans.
 - EJB 3.0 message-driven beans.
 - Servlets, but injection is not supported in JSPs.
 - The main class of the application client.
- **EntityManagerFactory:** The factory is used to create an EntityManager for database interactions.
- **Persistence unit:** A persistence unit consists of the declarative metadata that describes the relationship of entity class objects to a relational database. The EntityManagerFactory uses this data to create a persistence context that can be accessed through the EntityManager.
- **Persistence context:** The persistence context is the set of active instances that the application is handling. The persistence context can be created manually or through injection.

- Entity manager

The EntityManager interface is an API that manages the lifecycle of an entity instance.

- JPA query language

The Java persistence query language (JPQL) is used to define searches against persistent entities independent of the mechanism that is used to store those entities.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Entity manager

The EntityManager interface is an API that manages the lifecycle of an entity instance.

Entities cannot persist themselves on the relational database; annotations are used only to declare a POJO as an entity or to define its mapping and relationships with the corresponding tables on the relational database.

In JPA, the EntityManager interface is used to allow applications to manage and search for entities in the relational database.

The EntityManager is an API that manages the lifecycle of entity instances. An EntityManager object manages a set of entities that are defined by a persistence unit. Each EntityManager instance is associated with a *persistence context*. A persistence context defines the scope under which particular entity instances are created, persisted, and removed through the APIs made available by EntityManager. In some ways, a persistence context is conceptually similar to a transaction context.

The entity manager tracks all entity objects within a persistence context for changes and updates that are made, and flushes these changes to the database. Once a persistence context is closed, all managed entity object instances become detached from the persistence context and its associated entity manager, and are no longer managed. Once an object is detached from a persistence context, it can no longer be managed by an entity manager, and any state changes to this object instance will not be synchronized with the database.

Managed and unmanaged entities

An entity object instance is either managed (attached) by an entity manager or unmanaged (detached).

When an entity is attached to an entity manager, the manager monitors any changes to the entity and synchronizes them with the database whenever the entity manager decides to flush its state.

When an entity is detached, and therefore is no more associated with a persistence context, it is unmanaged, and its state changes are not tracked by the entity manager.

Entity instances become unmanaged and detached when a transaction scope or extended persistence context ends. An important consequence of this fact is that detached entities can be serialized and sent across the network to a remote client. The client can make changes remotely to these serialized object instances and send them back to the server to be merged back and synchronized with the database.

Note: This behavior is different from the EJB 2.1 entity model, where entities are always managed by the container.

Because in EJB 3.0 you are working with entities that are POJOs, this can simplify how you design Java™ EE applications, because you are not forced to use patterns, such as data transfer objects (DTO), between the business logic layer (session beans) and the persistence layer.

Entity manager operations

The main operations that can be performed by an entity manager: *Table 1. Entity manager operations. The table describes entity manager operations.*

Operation	Description
persist	Insert a new entity instance into the database. Save the persistent state of the entity and any owned relationship references. The entity instance becomes managed.
find	Obtain a managed entity instance with a given persistent identity (primary key), return null if not found.
remove	Delete a managed entity with the given persistent identity from the database.
merge	State of a detached entity gets merged into a managed copy of the detached entity. The managed entity that is returned has a different Java identity than the detached entity.

refresh	Reload the entity state from the database.
lock	Set the lock mode for an entity object that is contained in the persistence context.
flush	Force synchronization with database.
contains	Determine if an entity is contained by the current persistence context.
createQuery	Create a query instance using dynamic Java Persistent Query Language.
createNamedQuery	Create an instance of a predefined query
createNativeQuery	Create an instance of an SQL query.

Container-managed entity manager

One way to use an entity manager in a Java EE environment is with a container-managed entity manager. In this mode, the container is responsible for the opening and closing of the entity manager and thus, the lifecycle of the persistence context (in a way that is transparent to the application). A container-managed entity manager is also responsible for transaction boundaries.

A container-managed entity manager is obtained in an application through dependency injection or through JNDI lookup, and the container manages interaction with the entity manager factory transparently to the application.

A container-managed entity manager requires the use of a JTA transaction, because its persistence context is automatically propagated with the current JTA transaction, and the entity manager references that are mapped to the same persistence unit provides access to this same persistence context within the JTA transaction. This propagation of persistence context by the Java EE container means that the application does not have to pass references to the entity manager instances from one component to another.

Container-managed persistence contexts might be defined to have one of two different scopes:

- Transaction persistence scope
- Extended persistence scope

Application-managed entity manager

Using an application-managed entity manager allows you to control the entity manager in application code.

When using such an application-managed entity manager, note the following things:

- With application-managed entity managers the persistence context is not propagated to application components, and the lifecycle of entity manager instances is managed by the application. This means that the persistence context is not propagated with the JTA transaction across entity manager instances in a particular persistence unit.
- The entity manager, and its associated persistence context, is created and destroyed explicitly by the application.

This type of entity manager is used in two different scenarios:

- In Java SE environments, where you want to access a persistence context that is stand-alone, and not propagated along with the JTA transaction across the entity manager references for the given persistence unit.
- Inside a Java EE container, when you want to gain very fine-grained control over the entity manager lifecycle.

Parent topic: [JPA architecture](#)

JPA query language

The Java™ persistence query language (JPQL) is used to define searches against persistent entities independent of the mechanism that is used to store those entities.

JPQL is therefore portable, and not constrained to any particular data store.

JPQL is an extension of the Enterprise JavaBeans query language, EJB QL, and is designed to combine the syntax and simple query semantics of SQL with the expressiveness of an object-oriented expression language.

JPQL works with JPA elements in the following way:

- The application creates an instance of the [javax.persistence.EntityManager](#) interface.
- The EntityManager creates an instance of the [javax.persistence.Query](#) interface, through its public methods, for example `createNamedQuery`.
- The Query instance runs a query (to read or update entities).

Query instances are created using the methods that are exposed by the EntityManager interface.

Named queries

JPQL defines two types of queries: dynamic queries, which are created on the fly, and named queries.

Named queries are intended to be used in contexts where the same query is started several times. Their main benefits include the improved reusability of the code, a minor maintenance effort, and potentially better performance, because they are evaluated once.

Named queries are defined using the `@NamedQuery` annotation. The `name` attribute is used to uniquely identify the named query, while the `query` attribute defines the query.

Parent topic: [JPA architecture](#)

Creating JPA projects

You can use the JPA Project wizard to create and configure a JPA project.

About this task

To create a JPA project:

Procedure

1. In the JPA perspective, click **File > New > Project > JPA Project**.
2. In the **Project Name** field, type a name for the new project.
3. To change the default project location, clear the **Use default** check box under **Project contents** and select a new location with the **Browse** button.
4. In the **Target runtime** field, select the target runtime environment for the project. This field is disabled if you previously created a project and set a target runtime environment.
5. Select the JPA version to use in the project.
6. Under **Configurations**, select **Minimal Configuration**.
7. Optional: To add the new project to an enterprise application (EAR) file, select the **Add module to an EAR application** check box. Type a new project name or select an existing enterprise application project from the **EAR Project Name** list. Or, click **New** to start the New EAR Application Project wizard.**Note:** If you type a new EAR project name, the EAR project is created in the default location with the lowest compatible J2EE version based on the version of the project being created. If you want to specify a different version or a different location for the enterprise application, you must use the New Enterprise Application Project wizard.
8. Click **Next**.
9. Optional: Select source folders on the build path or add more by clicking **Add Folder**, set the default output folder, or leave the default. Click **Next**.
10. On the JPA facet page, do the following steps:
 - A. Select the proper platform.
 - B. Select the Library Provided by Target Runtime type in JPA implementation.
 - C. Select the appropriate database connection for this project, or click **Add Connection** to create a new one.**Note:** You must have an active database connection to map entities to database tables or to generate JPA entities from an existing database.
 - D. For **Persistent class management**, select whether you want annotated classes to be discovered automatically by the runtime environment or if they must be listed in the `persistence.xml` file.
 - E. If you want to automatically create the file `orm.xml`, select **Create orm.xml**. You can use the `orm.xml` file to override object-relational mapping details that are specified in Java™ classes through JPA annotations. The `orm.xml` file is in the META-INF directory for the project.
11. Click **Finish**.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Adding JPA support to a project

You can add Java™ Persistence API (JPA) support to a faceted project by adding the Java Persistence project facet.

Before you begin

You can only add the JPA project facet to faceted projects - projects that use the faceted project framework for adding functionality. Examples of faceted projects are web projects and an EJB projects. A project facet is a specific unit of functionality that you can add to a project when that functionality is required. When a project facet is added to a project, it can add natures, builders, classpath entries, and resources to a project, depending on the characteristics of the particular project. **Note:** Java projects (**File > New > Java Project**) are not faceted projects by default. To convert a Java project to a JPA project, see [Converting a Java project to a JPA project](#).

About this task

To add JPA support to a project:

Procedure

1. In the Package Explorer view, select the project and then in the main menu click **Project > Properties**. The Properties page opens.
2. Click the **Project Facets** property.
3. In the list of project facets, select **JPA** and then, in the Version column, select the version and then click **Apply**. The JPA facet is added to your project.
4. Click **OK**. The properties page closes.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Converting a Java project to a JPA project

You can convert a plain Java™ project to a JPA-enabled project.

About this task

To convert a Java project to a JPA project:

Procedure

1. Open a Java source file containing a class that you want to convert to a JPA entity bean.
2. In the Java editor, add one of the following JPA annotations immediately before the class declaration:
 - @Entity
 - @Embeddable
 - @Mappedsuperclass
3. You can see a Quick Fix (light bulb) icon next to the annotation that you typed. Click the **Quick Fix** icon or press **Ctrl+1** to view the suggestion for how to correct the problem.
4. Click the **Add JPA support** suggestion. The Java project is converted to a JPA project, the appropriate import statement is added to the class file for the annotation that you typed, and the class file is added to the `persistence.xml` file.
5. Press **Ctrl+S** to save the class file.
6. Switch to the JPA Development perspective (**Window > Open Perspective > JPA Development**), finish creating the new JPA entity bean, and create additional entities as needed.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Changing a JPA 1.0 project to JPA 2.0

You can change a JPA 1.0 faceted project to JPA 2.0

About this task

Complete the following steps to migrate a faceted project from JPA 1.0 to JPA 2.0.

Procedure

1. Right-click the project and click **Properties**.
2. Click **Project Facets**.
3. Locate JPA in the Project Facets list. In the Version column, click the version and select **2.0** from the list.
4. Click the **Further configuration required** link. The Modify Faceted Project window opens.
5. Select a Platform from the list that supports the new facet version.
6. Under JPA Implementation, select **Library Provided by Target Runtime** and then click **OK**.
7. Click **OK** to close the Project Facets window.
8. In the Enterprise Explorer view or another similar view, locate the `persistence.xml` file. Right-click the file and click **Upgrade document version**. The `persistence.xml` file is upgraded to version 2.0. Confirm the upgrade by opening the file in the Persistence XML editor and check that the version is 2.0.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Creating JPA entity beans

You can create JPA entity beans from a wizard, by adding persistence to a POJO, or from existing database tables.

- **Creating a JPA entity bean using a wizard**

You can create JPA entity beans using the New Entity wizard.

- **Creating a JPA entity bean by adding persistence to a POJO**

With the JPA Tools, you can create a JPA entity bean and add persistence to plain old Java objects (POJOs).

- **Creating JPA entity beans in a JPA project from database tables**

You can generate JPA entity beans from existing database tables (bottom-up mapping).

- **Creating JPA entity beans in a JPA-enabled web project from existing database tables**

You can create JPA entity beans for a JPA-enabled web project by generating the beans from existing database tables.

- **Generating database tables from entity beans (top-down mapping)**

Using the JPA tools, you can generate data definition language (DDL) files for creating database tables from entity beans that you create.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Creating a JPA entity bean using a wizard

You can create JPA entity beans using the New Entity wizard.

Before you begin

If you have not done so already, create a JPA project or enable JPA support in an appropriate project.

About this task

To create a JPA entity bean:

Procedure

1. Open the JPA perspective (**Window > Open Perspective > Other > JPA**).
2. In the Package Explorer view, right-click the project that you want to add the entity to and click **New > JPA Entity**.
3. Ensure that the correct Project to add your entities to is selected in the **Project** field.
4. In the **Source folder** field, type or browse to the path to the folder that contains the Java™ source file for your entity.
5. In the Java package field, type or browse to the Java package for your entities.
6. Type the class name of the entity in the corresponding field.
7. Optional: If your entity is derived from a superclass, type the name of superclass in the corresponding field.
8. Under Inheritance, select
 - Entity (this is selected by default)
 - Mapped Superclass
 - Inheritance
9. To add the entity mappings to an XML mapping file (usually META-INF/orm.xml), click **Add to entity mappings in XML**.
10. Click **Next**.
11. Type the name of your entity. By default it is pre-filled with the class name.
12. Under Table Name, type the name of the database table your entity maps to. By default, the table name is the same as your class name.
13. Add fields to your entity. To add field, click **Add** under Entity Fields and do the following steps:
 - A. Select the type.
 - B. Type the field name.
14. In the Key column of the Entity Fields table, click one or more fields that are the primary key for your entity.
15. Select if the access type for the entity is field- based or property-based.
16. Click **Next**.
17. Optionally, add the entity class to a class diagram for visualization.
18. Click **Finish**.

Results

The JPA entity is created for your project.

What to do next

You can further configure your entity (for example, by adding relationship mappings) by using the following tools:

- The Configure JPA Entities wizard. (Right-click the entity and click **JPA Tools > Configure JPA Entities**).
- The JPA Details view and the JPA structure view.

Parent topic: [Creating JPA entity beans](#)

Creating a JPA entity bean by adding persistence to a POJO

With the JPA Tools, you can create a JPA entity bean and add persistence to plain old Java™ objects (POJOs).

Procedure

1. Create a JPA project or enable JPA support in an appropriate project. Ensure that an active database connection is defined for the project.
2. Open the JPA perspective. (Click **Window** > **Open Perspective** > **Other** > **JPA**.)
3. Create a Java class (**File** > **New** > **Class**).
4. In the Package Explorer view, right-click on the new class and select **JPA Tools** > **Make Persistence**. You can select **Annotate** in Java and List in `persistence.xml`. Click **Next**. Specify the database table to map and click **Next** to specify the mapping of the bean attributes to database columns. Click **Finish**.
5. The new class now appears in the JPA structure view. Click it to go to the JPA Details view. To change the mapping type, click the type that the class is mapped to. You can add the following types of persistence to the class:
 - [Entity](#)
 - [Embeddable](#)
 - [Mapped Superclass](#)
6. If you selected **entity** in the previous step, you can [further configure the properties of the entity using the JPA details view](#).
7. For each attribute in your persistent class, you can specify the mapping of the attribute. In this step, you define how each class attribute maps to the database. In the JPA Structure view, click the attribute that you want to map, and then in the JPA Details view, click the **Map As** drop-down list. You can choose the following types of mappings (see the Java Persistence Tools User Guide for details on completing the fields):
 - [Basic mapping](#)
 - [Embedded mapping](#)
 - [Embedded ID mapping](#)
 - [ID mapping](#)
 - [Many-to-many mapping](#)
 - [Many-to-one mapping](#)
 - [One-to-many mapping](#)
 - [One-to-one mapping](#)
 - [Transient mapping](#)
 - [Version mapping](#)

Note: Entity (`@Entity`) persistent classes must have one attribute with an ID mapping.

Parent topic: [Creating JPA entity beans](#)

Creating JPA entity beans in a JPA project from database tables

You can generate JPA entity beans from existing database tables (bottom-up mapping).

Before you begin

Create a JPA project or enable JPA support in an appropriate project.

About this task

To generate entity beans from tables in a database:

Procedure

1. In the Package Explorer view, right-click the JPA project and select **JPA Tools > Generate Entities from Tables**.
2. On the Generate Custom Entities page, select a database connection and schema. If you have not created the database connection, click the **Add connections** icon and follow the prompts in New Connection Profile wizard to complete the new connection.
3. Select the tables from which you want to generate JPA entities.
4. Create, edit, or delete table associations by using the proper icons. Click **Next**.
5. Optionally, you can customize aspects of the entities to be generated, change the package and class, and add interfaces. Click **Next**.
6. In the Customize Individual Entities page, you can select tables and columns and set up the mapping values for each one. Click **Finish**. The entities are generated.
7. Add a primary key (@Id annotation) to the entities:
 - A. In the Package Explorer view, right-click one of the entities and select **JPA Tools > Configure JPA Entities**
 - B. In the Configure JPA Entities wizard, select the entities that you created and then click **Next**.
 - C. Click **Primary Key**.
 - D. Click the entity to configure and then select the attribute to set as the primary key (@Id) for the entity.
 - E. Repeat the previous step for any additional entities that need a primary key set.
8. Optional: Add additional configuration details to the entities before exiting this wizard.
9. Click **Finish**.

Parent topic: [Creating JPA entity beans](#)

Creating JPA entity beans in a JPA-enabled web project from existing database tables

You can create JPA entity beans for a JPA-enabled web project by generating the beans from existing database tables.

Before you begin

[Create a JPA-enabled web project.](#)

Procedure

1. In Enterprise Explorer, right-click your JPA-enabled web project and select **JPA Tools > Generate Entities From Tables**. The Generate Custom Entities wizard opens.
2. On the Select Tables page of the wizard, specify the connection name and the connection schema.
3. In the Tables list, select the tables for which you want to generate entities.
4. Click **List generated classes** in `persistence.xml`. This option enables automatic synchronization of the `persistence.xml` with any JPA resources in your workspace.
5. Click **Next**.
6. On the Table Associations page of the wizard, click the + button to create an association. The Create New Association wizard opens.
7. [Create an association](#). Click **Finish** to exit the Create New Association wizard.
8. [Customize the default entity generation](#) and click **Next**.
9. [Customize individual entities](#).
10. Click **Finish**.

Results

The JPA entity beans are added to your web project. You can view them in the JPA Content node.

Parent topic: [Creating JPA entity beans](#)

Generating database tables from entity beans (top-down mapping)

Using the JPA tools, you can generate data definition language (DDL) files for creating database tables from entity beans that you create.

About this task

Important: This capability is supported only for projects that target WebSphere® Application Server for Version 7.0 feature packs, Version 8.0, and Version 8.5.

In top-down mapping, you start with entity beans and use them to create your database tables. You start from scratch with the entity definitions and the object-relational mappings, and then you derive database schemas from that data. If you use this approach, you are most likely concerned with creating the architecture of your object model and then writing your entity classes. These entity classes eventually drive the creation of your database model. If you are using a top-down mapping of the object model to the relational model, develop the entity classes, and then use the JPA tools DDL generation capability to create the database tables that are based on the entity classes.

The process of mapping database tables top-down from entity beans requires these steps:

Procedure

1. [Create a JPA project](#) or enable JPA support in an appropriate project.
2. [Create entity beans](#)
3. [Generate a data definition language \(DDL\) file using the JPA tools.](#)
4. Create a database connection for the JPA project.
5. Generate database tables from the DDL file.

- [Generating a data definition language file from a JPA project](#)

You can generate a data definition language (DDL) file for your JPA project.

Parent topic: [Creating JPA entity beans](#)

Related tasks:

[Adding JPA support to a project](#)

Generating a data definition language file from a JPA project

You can generate a data definition language (DDL) file for your JPA project.

About this task

Important: This capability is supported only for projects that target WebSphere® Application Server for Version 7.0 feature packs, Version 8.0, and Version 8.5.

To generate a DDL file for a JPA project:

Procedure

1. In the Package Explorer view, right-click the JPA project and select **JPA Tools > Generate Tables from Entities....**
2. The data definition language file is created in the `META-INF` directory of your JPA project. Select the database connection and specify the file name and the schema. Click **Finish**.

What to do next

You can use the Table.ddl file to generate tables for your entity beans in a database. To generate the tables:

1. Right-click the Table.ddl file and click **Open With > SQL File Editor**.
2. In the SQL File Editor, enter the connection profile information `type`, `name`, and `database`.
3. In the editor, right-click and select **Execute All**.
4. Check the SQL Results view to verify that the table generation is successful.

Parent topic: [Generating database tables from entity beans \(top-down mapping\)](#)

Configuring JPA entity beans

Before you begin

1. [Create a JPA-enabled web project.](#)
2. [Create a JPA entity bean.](#)

Procedure

1. In Package Explorer, right-click the JPA project and select **JPA Tools** > **Configure JPA Entities**. The Configure JPA Entities wizard opens.
2. In the Available JPA Entities list, select the JPA entity beans that you want to configure or click [Create New JPA Entities](#) to create new entities. Click **Next**.
3. For each entity bean, you can configure:
 - **Primary Key**
 - For each JPA entity bean, specify the primary key. The primary key uniquely identifies each record in a set of data.
 - **Relationships**
 - Add new relationships between JPA entities or delete relationships.
 - **Named Queries**
 - Add new queries, edit existing queries, or delete queries associated with a JPA entity. Named queries are used for querying and retrieving data from the database.
 - **Concurrency Control**
 - Specify the optimistic lock value for the entity. The optimistic lock value ensures that correct results for concurrent operations are generated.
 - **Other**
 - Set advanced options.
4. After you configure the entity bean, click **Finish**.

- [Configuring the primary key](#)
- [Configuring named queries](#)
- [Configuring relationships](#)
- [Configuring concurrency control](#)
- [Configuring advanced options](#)

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Configuring the primary key Procedure

1. On the Tasks page of the wizard, select **Primary Key** from the Task list.
2. In the Attributes list, select an attribute or attributes that uniquely identifies an entity bean. For example, a department number might be a unique attribute for a department bean.

Parent topic: [Configuring JPA entity beans](#)

Configuring named queries

Procedure

1. On the Tasks page of the wizard, select **Named Queries** from the Task list.
2. The queries list displays all of the generated queries. You can edit the existing queries, remove the queries, and create your own queries.
3. To add a new query, click **Add** to open the Add Named Query window.
4. On the Result Attributes tab, select the attributes for the query to return, or select a result bean to map to the entity attributes.
5. In the **Named Query Name** field, enter a name to uniquely identify the named query. The text area displays the query as you are building it.
6. On the Filter Results tab, define filters for the query.
7. On the Order Results tab, define how the query results are displayed.
8. Click **OK** to close the Add Named Query window.

Parent topic: [Configuring JPA entity beans](#)

Configuring relationships

Procedure

1. On the Tasks page of the wizard, select **Relationships** from the Task list.
2. A relationship is a reference from one entity to another. To add a relationship, click **Add**.
3. Select the entity with which you want to create the relationship.
4. Select the fetch type. A lazy fetch type specifies that loading of the field is deferred until you access the field for the first time. An eager fetch type specifies that when you retrieve an entity, you are guaranteed that all of its fields are populated with data store data. For example, if you have a bidirectional one-to-many relationship between a Department entity and an Employee entity, since most of the time, when the Department entity is fetched, its employees are likely to be fetched, the fetch type of this one-to-many relationship is set to EAGER.
5. Select the multiplicity of the relationship.
6. Specify if you want the relationship to be bidirectional or unidirectional. A unidirectional relationship means that entity A references entity B, but entity B does not reference entity A. A bidirectional relationship means that the entities reference each other.
7. Select foreign keys from the list to map to primary keys.

Parent topic: [Configuring JPA entity beans](#)

Configuring concurrency control Procedure

1. On the Tasks page of the wizard, select **Concurrency Control** from the Task list. Concurrency control helps prevent lost updates, where two users update the same data and one of the updates overwrites the other update.
2. Specify how you want to manage the contention for data resources.

Parent topic: [Configuring JPA entity beans](#)

Configuring advanced options

About this task

Note: Not all of the options are available in all scenarios.

To configure advanced options:

Procedure

1. On the Tasks page of the wizard, select **Other** from the Task list.
2. Select **Add an equals and hashCode method if not present** to enable you to add and remove objects from collections.
3. Select **Convert Java Set objects to Java List objects** to convert `java.util.Set` to `java.util.List`.
4. Select **Convert Java SQL Temporal types to Java Util temporal types** to convert `java.sql.Date` types to `java.util.Date` types.

Parent topic: [Configuring JPA entity beans](#)

Adding a primary key to the JPA entity

Every entity that is mapped to a relational database must have a mapping to a primary key in the table.

About this task

The EJB 3.0 specification supports three different ways to specify the entity identity:

- @Id annotation
- @IDClass annotation
- @EmbeddedId annotation

- ID annotation

The @Id annotation offer the simplest mechanism to define the mapping to the primary key.

- IDclass annotation

The @IDClass annotation is used to model a composite key.

- Embedded ID annotation

Use the @EmbeddedId annotation with the @Embeddable annotation to move the definition of a composite key inside the entity.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

ID annotation

The `@Id` annotation offer the simplest mechanism to define the mapping to the primary key.

You can associate the `@Id` annotation to fields/properties of these types:

- Primitive Java™ types and their wrapper classes
- Arrays of primitive or wrapper types
- Strings: `java.lang.String`
- Large numeric types: `java.math.BigDecimal`, `java.math.BigInteger`
- Temporal types: `java.util.Date`, `java.sql.Date`

We discourage the usage of floating point types (float and double, and their wrapper classes) for decimal data, because you can have rounding errors and the result of equals operator is unreliable in these cases. Use `BigDecimal` instead.

The `@Id` annotation fits well in scenarios where a natural primary key is available, or when database designers use surrogate primary keys (typically an integer) that has no descriptive value and is not derived from any application data in the database. On the other end, composite keys are useful when the primary key of the corresponding database table consists of more than one column. Composite keys can be defined by the `@IdClass` or `@EmbeddedId` annotation.

Parent topic: [Adding a primary key to the JPA entity](#)

IDclass annotation

The @IDClass annotation is used to model a composite key.

Parent topic: [Adding a primary key to the JPA entity](#)

Embedded ID annotation

Use the `@EmbeddedId` annotation with the `@Embeddable` annotation to move the definition of a composite key inside the entity.

The `@Embeddable` annotation is used to model persistent objects that have no identity of their own, because they are nested inside another entity.

Parent topic: [Adding a primary key to the JPA entity](#)

Synchronizing persistent classes in the persistence.xml file

After you create persistent classes, you can automatically add them to list of classes in their persistence unit in the persistence.xml file.

About this task

The persistence.xml file is in the META-INF directory of the JPA project. To automatically add all persistent classes in your JPA project to the persistence.xml file:

Procedure

1. In the Package Explorer view, right-click the persistence.xml file.
2. Select **JPA Tools > Synchronize Class List**.

Results

The persistent classes in your JPA project are automatically discovered and added to the persistence unit in the persistence.xml file.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Working with persistence units (persistence.xml)

You can edit the `persistence.xml` file with the Persistence XML Editor.

About this task

The `persistence.xml` file describes the details of the persistence units in your JPA project. A persistence unit contains a list of entity beans.

The Persistence XML editor simplifies making changes to the `persistence.xml` file:

- Add and remove persistence units.
- Modify the properties of a persistence unit.
- Add and remove classes in a persistence unit.
- Add and remove properties in a persistence unit.

The `persistence.xml` file is in the `META-INF` directory of the JPA project.

To edit the `persistence.xml` file:

Procedure

1. In the Package Explorer view, right-click the `persistence.xml` file of the JPA project that you want to edit and select **Open with > Persistence XML Editor**.
2. In the Design tab of the Persistence XML Editor, make any of the following changes to the `persistence.xml` file:
 - Add or remove persistence units.
 - [Modify the persistence unit details](#).
 - Add or remove classes for a persistence unit.
 - Add or remove additional properties of a persistence unit.
3. Click **File > Save** to save your changes.

- [Modifying the persistence unit details](#)

Use the Persistence XML editor to modify the properties of your persistence units.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Modifying the persistence unit details

Use the Persistence XML editor to modify the properties of your persistence units.

Procedure

1. Open the `persistence.xml` file in the Persistence XML editor.
2. Click the persistence unit.
3. Modify the properties of the persistence unit as described in the following table: *Table 1. Persistence unit details*

Attribute	Description
Name	The name of the persistence unit. The name attribute is required.
Description	A description of the persistence unit.
Provider	Name of the <code>javax.persistence.spi.PersistenceProvider</code> class interface for the persistence provider.
Transaction Type	Choose if it is a Java™ transaction API (JTA) or a non-JTA data source.
JTA Data Source	Global JNDI name of the JTA data source.
Non-JTA data source	Global JNDI name of a non-JNDI data source.
Exclude unlisted classes	False, by default. When set to True, classes that are not listed in the <code>persistence.xml</code> file are excluded from the persistence unit.
JAR File	The names of JAR files containing entities.
Mapping File	If you use a custom mapping file, list it here. (If you use the <code>orm.xml</code> file for mapping, it is read automatically.)

Parent topic: [Working with persistence units \(persistence.xml\)](#)

Adding or overriding entity mappings in the `orm.xml` file

You can use the Object Relational Mapping XML Editor to define object-relational mappings for JPA entity beans in the `orm.xml` file.

About this task

When you specify object-relational mappings using the JPA tools, the mapping information is contained in the Java™ class files in the form of Java annotations. However, you can also choose to define the object-relational mappings in XML in a file called `orm.xml`.

Mapping information that is defined in the `orm.xml` file automatically overrides both default JPA behavior and any mappings defined using annotations. Therefore, you can, for example, to adapt existing JPA entity beans to a different set of database tables without needing to modify the entity class files.

To edit the `orm.xml` file that is in the `META-INF` directory of the JPA project:

Procedure

1. In the Package Explorer view, right-click the `orm.xml` file of the JPA project that you want to edit, and select **Open with** > **Object Relational Mapping XML Editor**.
2. In the Design tab of the Object Relational Mapping XML Editor, add object-relational mapping information.
3. Click **File** > **Save** to save your changes.

Parent topic: [Developing Java Persistence API \(JPA\) applications](#)

Developing WebSocket applications

WebSocket technology improves upon the existing HTTP request-response model, enables real-time communication, and improves performance and scalability for applications that require timely updates from the server. WebSocket applications consist of WebSocket endpoints, which can be client or server implementations of business logic.

About this task

Important: Applicable editions: Liberty profile, Liberty Core

- **Creating WebSocket endpoints**

You can use wizards to create WebSocket endpoints in your workspace.

- **Deploying WebSocket applications**

You can deploy your WebSocket application using WebSphere® Application Server.

Parent topic: [Developing enterprise applications](#)

Related information:

[Liberty profile: WebSockets](#)

Creating WebSocket endpoints

You can use wizards to create WebSocket endpoints in your workspace.

Before you begin

Important: Applicable editions: Liberty profile, Liberty Core

- You can use the wizard to create the following endpoint types on Web/Web Fragment 3.1 and Utility projects:
 - Programmatic endpoint. Creating an endpoint with this option generates a new class, which extends the `javax.WebSocket.Endpoint` class and the `onClose`, `onOpen`, `onError` method stubs that you select.
 - Annotated client endpoint. Choosing this option generates a new class that is annotated with the `@ClientEndpoint` annotation.
 - Annotated server endpoint. You must specify the mandatory URL to complete the wizard. Choosing this option generates a new class that is annotated with the `@ServerEndpoint` annotation.

Procedure

1. Create a Web, Web Fragment, or Utility project.
2. Create a WebSocket Endpoint by using the file or context menu options:
 - Click **File > New > Other**. Under the Web category, select **WebSocket Endpoint**.
 - Right-click a selected project. Select **New > WebSocket Endpoint**.

Parent topic: [Developing WebSocket applications](#)

Deploying WebSocket applications

You can deploy your WebSocket application using WebSphere® Application Server.

Before you begin

Important: Applicable editions: Liberty profile, Liberty Core

When you deploy on an Enterprise Archive (EAR) file, all projects defined in the EAR file are deployed.

Procedure

1. Deploy your WebSocket application to WebSphere Application Server Version 8.5.5.4 and later.
2. Package your WebSocket applications as part of a Web application Archive (WAR) file or as a Java Archive (JAR) file under the `WEB-INF/lib` directory. To decrease the time required for deployment, you can provide a **`ServerApplicationConfig`** implementation to skip the scanning of JAR files that are in the `WEB-INF/lib` directory and that do not have WebSocket applications.

Parent topic: [Developing WebSocket applications](#)

Developing applications that use Contexts and Dependency Injection (CDI)

You can use wizards to create applications that use Contexts and Dependency Injection (CDI).

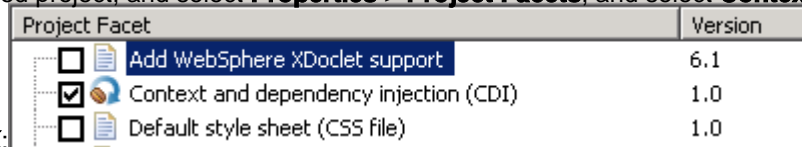
Before you begin

Create a Java™ EE-faceted project (that is, a utility, EJB, web, connector, or application client project) in your workspace.

For information about creating one of these projects see: [Creating and configuring Java EE projects using wizards](#).

Procedure

1. Right-click your Java EE-faceted project, and select **Properties > Project Facets**, and select **Context and dependency**



injection, click **Apply** and **OK**:

To see the `beans.xml` file, expand:

- Utility projects: `<proj>/src/META-INF`
- EJB projects: `<proj>/ejbModule/META-INF`
- WEB projects: `<proj>/WebContent/WEB-INF`
- Connector projects: `<proj>/connectorModule/META-INF`
- Application Client project: `<proj>/appClientModule/META-INF`

At this point, it is empty: `<?xml version="1.0" encoding="UTF-8"?>`

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
</beans>
```

2. You now have a CDI-enabled project.

- [Learn about Contexts and Dependency Injection \(CDI\)](#)

Contexts and dependency injection (CDI) for the Java EE platform is an implementation based on the JSR 346 specification. You can create applications that implement CDI in your Java EE projects.

- [Implied values in contexts and dependency injection annotations](#)

The workspace supports implied value support for the `@Named` and `@New` annotations.

- [Editing and validating your contexts and dependency injection projects](#)

You can use the contexts and dependency injection deployment descriptor editor to edit and validate your `beans.xml` file.

Parent topic: [Developing enterprise applications](#)

Learn about Contexts and Dependency Injection (CDI)

Contexts and dependency injection (CDI) for the Java™ EE platform is an implementation based on the JSR 346 specification. You can create applications that implement CDI in your Java EE projects.

Overview

CDI 1.0 applications are activated by the presence of a `beans.xml` file that exists in the `WEB-INF` directory of a web archive (WAR), or in the `META-INF` directory of other types of archives, as defined by the JSR 299 specification.

8.5.5.6 [Liberty profile] Starting with CDI 1.1 (JSR 346), the presence of a `beans.xml` file is no longer required if bean defining annotations are specified. For more information about bean defining annotations, see the [CDI 1.2 specification](#).

When activated, the container provides services such as:

- Context management
- Type-safe dependency injection: A CDI-managed bean is instantiated and injected as needed.
- Decorators, which implement one or more bean interfaces and can contain business logic. Decorators are disabled by default. You can have multiple decorators per bean and order is defined by the beans.
- Interceptor bindings. Interceptors, which are enabled manually in the `beans.xml` file, are bound using an interceptor binding type.
- Event model
- Integration into JavaServer Faces (JSF) and JavaServer Pages (JSP) files using the Expression Language (EL)

Validation for CDI applications

Your workspace provides in-line and quick-fix validations for contexts and dependency injection applications. In CDI-faceted projects, as-you-type validation is supported.

8.5.5.6 [Liberty profile] Starting with CDI 1.1, session beans or managed beans that are annotated with `@Vetoed` are considered disabled, and are not validated.

For more information about the CDI annotations, see the [Oracle Java EE API documentation](#).

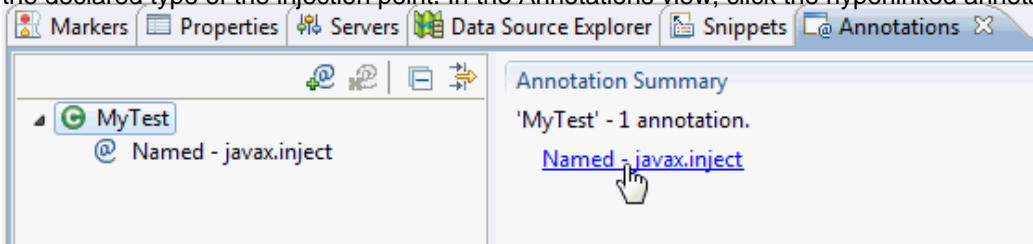
For more information about using CDI in WebSphere® Application Server, see [Contexts and Dependency Injection \(CDI\)](#).

Parent topic: [Developing applications that use Contexts and Dependency Injection \(CDI\)](#)

Implied values in contexts and dependency injection annotations

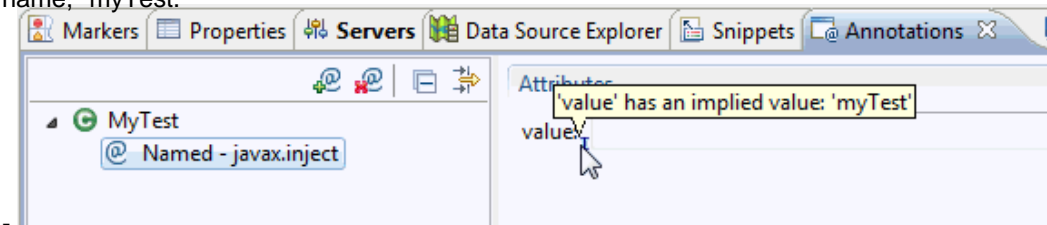
The workspace supports implied value support for the `@Named` and `@New` annotations.

When you specify an injection point (for example, `@Named`) and you do not specify a value, the implied value is derived from the declared type of the injection point. In the Annotations view, click the hyperlinked annotation to view its attributes:



Implied value for the `@Named` annotation

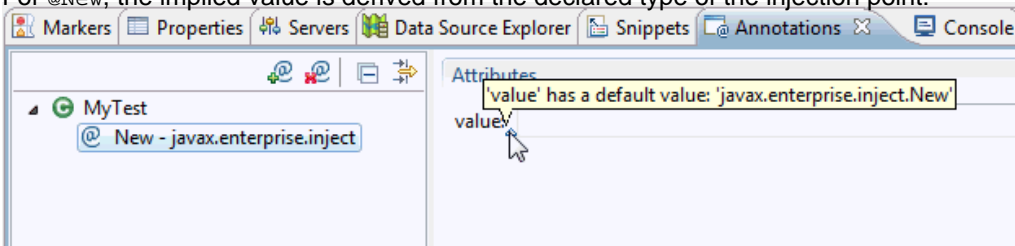
For `@Named`, the implied value is derived from the declared type of the injection point, the lowercase version of the class name, "myTest."



- For `Type`, it is the lowercase element name
- For Producer methods, it is the lowercase producer method name.
- For Producer fields, it is the field name.
- There is no implied value for all other cases.

Implied value for the `@New` annotation

For `@New`, the implied value is derived from the declared type of the injection point:



Parent topic: [Developing applications that use Contexts and Dependency Injection \(CDI\)](#)

Editing and validating your contexts and dependency injection projects

You can use the contexts and dependency injection deployment descriptor editor to edit and validate your `beans.xml` file.

Before you begin

Create a Java™ EE-faceted project (that is, a utility, EJB, web, connector, or application client project) in your workspace.

For information about creating one of these projects, see: [Creating and configuring Java EE projects using wizards](#).

About this task

In CDI 1.0, the `beans.xml` file is required in the packaging and deployment of a CDI-enabled project. The presence of the `beans.xml` file indicates to the container that the deployed module is a bean archive. The deployment descriptor is generated when you install the CDI facet. See [Creating applications that use Contexts and Dependency Injection \(CDI\)](#).

Initially, the `beans.xml` file is empty, which means that any annotated interceptors, decorators, or alternatives that exist in your module are disabled. To enable these classes, you can configure them using the CDI deployment descriptor editor.

8.5.5.6 [Liberty profile] Starting in CDI 1.2, the `beans.xml` file is optional in the packaging and deployment of a CDI-enabled project. This file is not created when you add the CDI 1.2 facet. You can generate this file by right-clicking on your project and selecting **Java EE Tools > Generate CDI Beans Deployment Descriptor Stub** from the menu.

Procedure

1. To open the context and dependency injection deployment descriptor editor, double-click your `beans.xml` file.
2. Click **Add** to add alternatives, decorators, and interceptors to your `beans.xml` file. **8.5.5.6** [Liberty profile] Starting in CDI 1.2, you can also click **Add** to specify `scan`, which you can use to set exclusion filters. Exclusion filters prevent classes from being discovered as beans.
3. **8.5.5.6** [Liberty profile] In the CDI 1.2 editor, use the following bean-discovery-mode values to specify how to scan your bean archive: `all`, `annotated`, or `none`. The CDI deployment descriptor editor issues validation errors under the following conditions:
 - If you have alternatives classes, decorators classes, or interceptors classes that are empty.
 - If you have alternatives classes, decorators classes, or interceptors classes that are non-existent.
 - If you have alternatives classes, decorators classes, or interceptors classes that are duplicates.
 - If you have classes that do not have the appropriate `@Alternative`, `@Interceptor`, or `@Decorator` annotations.

Parent topic: [Developing applications that use Contexts and Dependency Injection \(CDI\)](#)

Developing OSGi applications

OSGi applications are modular applications that you can manage without restarting your system. These topics provide instructions for creating, importing, exporting, and deploying OSGi applications.

- [Learn about OSGi applications](#)

The OSGi applications framework provides a programming model for developing, assembling, and deploying modular applications that use Java™ EE and OSGi technologies. OSGi application development tools provide a way to build enterprise applications that benefit from the modularity, dynamism, version control, and third-party library integration that is provided by the OSGi applications framework.

- [OSGi overview](#)

OSGi is a module system that is compatible with systems that are based on Java and implements a dynamic component model. Enterprise systems can use OSGi to improve the maintainability of runtime infrastructures. Applications, in the form of bundles, can be remotely installed, started, stopped, updated, and uninstalled without requiring a restart.

- [Tools for OSGi application development](#)

The OSGi application development tools help you to create OSGi applications.

- [Installing the server](#)

- [Creating OSGi projects](#)

OSGi projects hold all of the resources that you create, maintain, and use as you develop your applications.

- [Importing and configuring OSGi applications](#)

Learn how to import EBA files as OSGi applications, bundle JAR files as OSGi bundles, fragment JAR files as fragments, and CBA files as OSGi composite bundles. Learn also how to import deployment manifest files.

- [Exporting OSGi applications](#)

Learn how to export OSGi applications, OSGi bundles, OSGi fragments, and OSGi composite bundles. Learn also how to export deployment manifest files.

- [Converting existing projects to OSGi projects](#)

- [Adding OSGi support to Maven projects](#)

You can add OSGi support to Maven projects by converting the project to an OSGi bundle project.

- [Adding Maven support to OSGi applications](#)

You can add Maven support to OSGi applications by converting the project to a Maven project. The converted project remains an OSGi application but configurations for the bundle, such as its ID, version, and its dependencies are defined in a `pom.xml` file.

- [Accessing data using Java Persistence API \(JPA\)](#)

Java Persistence API (JPA) is a specification for the persistence of Java objects to relational databases. JPA helps you to manage relational data in your applications

- [Deploying OSGi applications](#)

Learn how to deploy your OSGi applications, and composite bundles.

- [Using independent Java archives \(JARs\) with OSGi applications](#)

- [Exposing EJBs as OSGi services](#)

You can include EJBs in OSGi bundles and expose EJBs as OSGi services.

- [Configuring OSGi bundle repositories](#)

- [OSGi application samples and tutorials](#)

To view the sample and tutorials in this product, click **Help > Help Contents** and expand the Samples and Tutorials sections. Learn about different aspects of OSGi application development from the following samples and tutorials:

Learn about OSGi applications

The OSGi applications framework provides a programming model for developing, assembling, and deploying modular applications that use Java™ EE and OSGi technologies. OSGi application development tools provide a way to build enterprise applications that benefit from the modularity, dynamism, version control, and third-party library integration that is provided by the OSGi applications framework.

- [Overview](#)
- [OSGi support in WebSphere Application Server](#)
- [Getting started](#)
- [Samples and tutorials](#)
- [Web resources for learning](#)

Overview

You can read the following topics before you create an OSGi application. They provide planning and technology overview information that might be useful if you are new to OSGi application development or developing OSGi applications in this development environment.

- [OSGi overview](#)
- [OSGi Blueprint Container](#)
- [Tools for OSGi application development](#)

OSGi support in WebSphere Application Server

You can develop the following Java EE applications as OSGi applications:

- Web applications
- Java Persistence API (JPA) applications
- JAX-RS (REST) Web services
- JavaServer Faces applications
- XML applications
- EJBs

Note: To develop OSGi applications for IBM® WebSphere® Application Server Version 7.0, you must install the Feature Pack for OSGi Applications and Java Persistence API 2.0. For information on installing WebSphere Application Server Version 7.0 Feature Pack for OSGi or WebSphere Application Server Version 8.0, refer to [Installing the server](#).

Table 1. WebSphere Application Server support for OSGi applications

	Version support in WebSphere Application Server version 7.0	Version support in WebSphere Application Server version 8.0	Version support in WebSphere Application Server version 8.5	Version support in Liberty profile server
Dynamic Web Modules	2.5	2.5 and 3.0	2.5 and 3.0	2.5 and 3.0
EJBs	Not supported	Not supported	3.0 and 3.1	Not supported
Java Persistence API	1.0 and 2.0	1.0 and 2.0	1.0 and 2.0	1.0 and 2.0
JavaServer Faces	1.1 and 1.2	1.1, 1.2, and 2.0	1.1, 1.2, and 2.0	Not supported
JAX-RS	1.1	1.1	1.1	1.1

SIP	1.0 and 1.1 Note: Annotated SIP1.1, or OSGi, applications are not supported	1.0 and 1.1 Note: Annotated SIP1.1, or OSGi, applications are not supported	1.0 and 1.1 Note: Annotated SIP1.1, or OSGi, applications are not supported	Not supported
-----	--	--	--	---------------

Restriction:The following technologies are not supported by the WebSphere Application Server Version 7.0 OSGi run time:

- EJB
- RPC Adapter
- Servlet 3.0
- Web 2.0 Server-side technologies
- Web 3.0
- Web services
- XML Transformations APIs

The following technologies are not supported by the WebSphere Application Server Version 8.0 OSGi run time:

- EJB
- RPC Adapter
- Web 2.0 Server-side technologies
- Web services
- XML Transformations APIs



Table 2. WebSphere Application Server Publish support for OSGi projects. The WebSphere Application Server Publish support for OSGi projects table lists the versions of WebSphere Application Server that support specific OSGi applications or bundles.

		Supported in WebSphere Application Server version 7.0	Supported in WebSphere Application Server version 8.0	Supported in WebSphere Application Server version 8.5	Supported in Liberty profile server
OSGi Applications containing:	OSGi Bundles	✓	✓	✓	✓
	OSGi Bundles with EJBs			✓	
	OSGi Fragments	✓	✓	✓	✓
	OSGi Composite bundles		✓	✓	
	Java EE WAR Modules	✓	✓	✓	
	PDE plug-ins	✓	✓	✓	✓
	PDE fragments	✓	✓	✓	✓
OSGi Composite bundles containing:	OSGi Bundles		✓	✓	

OSGi Bundles with EJBs			✓	
OSGi Fragments		✓	✓	
PDE plug-ins		✓	✓	
PDE fragments		✓	✓	

Getting started







If you are already familiar with OSGi development technology, the following topics guide you through the development process.

-  [Create an OSGi bundle project](#)
-  [Create an OSGi application project](#)
-  [Deploy the OSGi application](#)

Samples and tutorials




For a complete list of OSGi application development samples and tutorials, refer to [OSGi application samples and tutorials](#). To view the sample and tutorials in this product, click **Help > Help Contents** and expand the Samples and Tutorials sections.

Learn about different aspects of OSGi application development from the following samples and tutorials:

-  [Sample: OSGi Hello World](#)
 - This sample OSGi application contains a servlet that demonstrates the use of an activator.
-  [Sample: OSGi Counter Service](#)
 - This sample OSGi application consists of an OSGi web bundle that contains a servlet that accesses a service that is provided in another bundle project. This sample is an introduction to using OSGi application development tools.
-  [Sample: OSGi Blog](#)
 - This sample OSGi application demonstrates how to structure the API and implementation code into separate bundles. The OSGi application consists of an OSGi web bundle that contains servlets that access a JPA service that is provided in another bundle.
-  [Sample: EJB temperature converter](#)
 - This OSGi sample demonstrates an EJB configured as an OSGi bundle and exposed as a service.
-  [Tutorial: Develop a simple OSGi application](#)
 - This tutorial demonstrates how to create an OSGi application and run it on WebSphere Application Server. The OSGi application consists of an OSGi web bundle that contains a servlet that accesses a service that is provided in another bundle project. This tutorial is an introduction to using OSGi application development tools.
-  [Tutorial: OSGi EJB service](#)
 - This tutorial demonstrates how to create an OSGi application that exposes an EJB as a service. It demonstrates how to create OSGi bundles with EJB support, use the OSGi tools to manage EJB exports, and create a servlet that accesses the EJB as an OSGi service.

Web resources for learning

In addition to the information found in this information center, the following links provide extra learning material.

-  [Developing enterprise OSGi applications for WebSphere Application Server](#)
-  [Best practices for developing and working with OSGi applications](#)
-  [Innovations within reach: Are we ready for enterprise OSGi?](#)

Note: Latest developerWorks® articles and tutorials about OSGi applications

Parent topic: [Developing OSGi applications](#)

OSGi overview

OSGi is a module system that is compatible with systems that are based on Java™ and implements a dynamic component model. Enterprise systems can use OSGi to improve the maintainability of runtime infrastructures. Applications, in the form of bundles, can be remotely installed, started, stopped, updated, and uninstalled without requiring a restart.

- [OSGi features](#)
- [Benefits of OSGi](#)

- **The OSGi specification**

OSGi specifications are defined and maintained by the OSGi Alliance, an open standards organization. The specification outlines open standards for the management of voice, data, and multimedia wireless and wired networks. The OSGi Service Platform Specification defines an open common architecture for service delivery and management using bundles.

- **OSGi architecture**

The core part of the OSGi Service Platform defines a secure and managed service platform that is based on Java. This platform supports the deployment of extensible and downloadable applications that are known as bundles. The specification defines a security model, an application lifecycle management model, a service registry, an Execution environment, and Modules.

- **OSGi bundles**

An OSGi bundle is a Java archive file that contains Java code, resources, and a manifest that describes the bundle and its dependencies. The bundle is the unit of deployment for an application.

- **OSGi applications**

An OSGi application groups a set of bundles to provide a coherent business logic. The application can consist of different bundle types such as web enabled bundles and persistence enabled bundles.

- **OSGi fragments**

An OSGi fragment is a Java archive file with specific manifest headers that enable it to attach to a specified host bundle or specified host bundles to function. Fragments are treated as part of the host bundles. Relevant definitions of the fragment are merged with the host bundles definitions before the host is resolved, provided the information does not conflict. Fragment dependencies are resolved if possible. If the fragment dependencies cannot be resolved, the fragment does not attach to the host bundle. A fragment cannot have its own class loader or bundle activator. It cannot override the information present in the host bundles. Fragments extend bundles with resources, classes, and permitted headers, which enable you to customize your bundles.

- **OSGi composite bundles**

Composite bundles group bundles into aggregates to ensure consistent behavior. A composite bundle contains bundles or references to bundles outside of the workspace or target platform. A composite bundle ensures consistent behavior from a set of shared bundles at a specific version.

Parent topic: [Developing OSGi applications](#)

OSGi features

OSGi tools include the following major features:

- **Container for OSGi Blueprint components**

- The OSGi application framework includes the [Apache Software Foundation's Aries](#) open implementation of the OSGi Version 4.2 Blueprint component model that defines a standard dependency injection mechanism for Java components. The implementation is derived from the Spring Framework and extended for OSGi to declaratively register component interfaces as services in the OSGi service registry.

- **Model for assembling bundles**

- The OSGi tools include a model for assembling an application into a deployable unit. The unit can consist of multiple bundles and includes the metadata that describes the version and external location of the constituent bundles of the application.

- Runtime components

- The OSGi tools support the development of OSGi applications that run in an OSGi framework, exploiting enterprise Java technologies common in web applications and integration scenarios that include web application bundles, remote services integration, and JPA.

- Extensions

- The OSGi tools include extensions that go beyond the OSGi Enterprise Expert Group specifications to provide a complete integration of OSGi modularity with Java enterprise technologies. In particular it delivers support that includes but is not restricted to the following features:
 - Isolated enterprise applications that are composed of multiple, versioned bundles with dynamic lifecycle.
 - Declarative transactions and security for Blueprint components.
 - Container-managed JPA for Blueprint components.
 - Message-driven Blueprint components.
 - Configuration of resource references in module Blueprint Services.
 - Annotation-based Blueprint configuration.
 - Federation of lookup mechanisms between local JNDI and the OSGi service registry.
 - Fully declarative application metadata to enable reflection of an SCA component type definition.

Benefits of OSGi

OSGi modularity provides standard mechanisms to address the issues faced by Java EE applications. The OSGi framework provides the following benefits:

- Applications are portable, easier to re-engineer, and adaptable to changing requirements.
- The framework provides the declarative assembly and simplified unit test of the Spring Framework, but in a standardized form that is provided as part of the application server runtime rather than being a third-party library deployed as part of the application.
- The framework integrates with the Java EE programming model, giving you the option of deploying a web application as a set of versioned OSGi bundles with dynamic lifecycle.
- It supports administration of application bundle dependencies and versions, simplifying and standardizing third-party library integration.
- The framework provides isolation for enterprise applications that are composed of multiple, versioned bundles with dynamic lifecycles.
- It has a built-in bundle repository that can host common and versioned bundles that are shared between multiple applications so that each application does not deploy its own copy of each common library.
- OSGi applications can access external bundle repositories.
- The framework reinforces service-oriented design at the module level.
- OSGi applications can be composed of coarser-grained SCA assemblies.

The OSGi specification

OSGi specifications are defined and maintained by the OSGi Alliance, an open standards organization. The specification outlines open standards for the management of voice, data, and multimedia wireless and wired networks. The OSGi Service Platform Specification defines an open common architecture for service delivery and management using bundles. The OSGi Applications framework provides a programming model for developing, assembling, and deploying, as bundles, modular applications that use both Java™ EE and OSGi technologies.

OSGi Service Platform Specifications Version 4.2 bring the benefits of OSGi to the Java EE application developer. The OSGi Version 4.2 standard defines the Blueprint component model. This model defines how you can exploit OSGi modularity in your applications, in particular to help with third-party library integration and version control. The OSGi Applications framework in WebSphere® Application Server includes the following major features:

- It implements a set of application-centric enterprise OSGi technologies that are used by Java application components.
- It advocates and extends the use of the Blueprint component model.
- It builds on the Apache Software Foundation Aries project, which provides an open implementation of the Blueprint container.

For more information about the OSGi Applications framework in WebSphere Application Server, see the Feature Pack for OSGi Applications and JPA 2.0 documentation in the [WebSphere Application Server library](#).

For more information about the OSGi specification, see [OSGi Alliance Specifications](#).

- [OSGi Blueprint Container Specification](#)

Parent topic: [OSGi overview](#)

Enterprise OSGi

OSGi for Java enterprise applications is the focus of Version 4.2 of the OSGi specification.

Version 4.2 of the OSGi specification includes the definition of the Blueprint component model, a standardized version of the Spring Framework assembly model. The Blueprint component model describes how components can be wired together within a bundle and how configurations and dependencies are injected by a Blueprint component container in the runtime environment.

Components and the references they consume are declared in an XML module Blueprint file that is a standardization of the Spring application context. The file is extended for the OSGi environment so that components can be automatically published as services for the service registry, and references can be automatically resolved as services discovered from the service registry.

The Blueprint component model provides the simplicity of the Spring Framework, including its ability to form a unit test that is separated from the server environment. Blueprint standardizes the configuration metadata, and therefore brings governance to the specification of the component model.

OSGi Blueprint Container Specification

The OSGi Blueprint Container Specification defines a dependency injection framework for OSGi derived from the [Spring Dynamic Modules project](#). The specification defines a component model for OSGi based on the core Spring framework in which an OSGi bundle is augmented by an XML *module blueprint*. A module blueprint is a configuration file that describes how fine-grained components are wired together within the bundle. For more information about the OSGi Blueprint Container Specification, see the Compendium Specification on the [OSGi Alliance website](#).

Module components are managed by a *module context container*, a direct equivalent of the Spring application context container that injects configured dependencies into the components and manages their lifecycle. The format of the module blueprint is based on the Spring application configuration file. The significant evolution from the Spring framework is the unit of deployment, which is known as an OSGi bundle, and the integration with the OSGi service registry through the module blueprint. OSGi services that are exposed to clients of the bundle and OSGi services that are consumed by the bundle are declared in the module blueprint and registered with or retrieved from the OSGi service registry by the runtime module context container.

In a Blueprint application, a module component is a Java™ component in which the lifecycle is managed by a module context container. The module component configuration includes references to resources and components on which it depends. The module context container injects the configuration into the module component. Having the configuration injected into the component, rather than the component being dependent on external factories and services, makes it easier to test the component in isolation.

A module context container is a set of managed components that are assembled into an OSGi bundle. The module context is responsible for managing the lifecycle of the managed component that it contains and for the injection of the component configurations.

Learn more about OSGi Blueprint concepts:The OSGi Blueprint Container Specification defines the following concepts:

- Managed component

- A managed component is a Java component whose lifecycle is managed by a container and whose configuration, including references to resources and the other components it depends on, is injected into it by the container. Having the configuration is injected into a component, rather than the component being dependent on external factories and services, makes it easy to test the component in isolation.

- Module context

- A module context is the container of a set of managed components that are assembled into an OSGi bundle. It is responsible for managing the lifecycle of its managed components and injecting the components' configuration. This terminology is derived from the Spring framework's "application context" container and is a little clumsy, especially as the Blueprint specification uses the term "module context" as shorthand both for the "module context container" and "module context configuration file". In the EBA programming model, the term EBA container is introduced and means the same as module context container.

- Module blueprint

- A module blueprint is the declarative configuration that is associated with the set of managed components in an OSGi bundle that is processed by the module context container. In the Blueprint specification, a module blueprint takes the form of one or more XML module context configuration files for which the Blueprint specification defines an extensible XML schema.

- Managed bundle

- A managed bundle is an OSGi bundle that contains a module blueprint that describes the set of managed components in the managed bundle.

The declarative configuration in the module blueprint can also specify that certain managed components of the bundle must be exported as services in the OSGi service registry. In addition, it is possible to declare that the managed component of a bundle depends on a service or set of services that are obtained through the service registry, and for those

services to be injected into the managed component.

Overall, the OSGi Blueprint Container Specification describes an application architecture in which application modules are implemented as OSGi bundles with a module blueprint (the configuration information) and a runtime context that is created from that blueprint. Modules are peers, which interact through the service registry.

- **OSGi Blueprint XML files**

Blueprint XML files define and describe the various components of an application.

Parent topic: [The OSGi specification](#)

OSGi Blueprint XML files

Blueprint XML files define and describe the various components of an application.

The Blueprint XML file contains definitions of various component managers. The Blueprint Container specification defines four main component managers: a bean manager, a service manager, and two service reference managers. Each manager has a corresponding XML element that describes the manager properties. Each manager is responsible for creating and managing the lifecycle of the components they create, providing a component instance.

The blueprint files for a bundle are XML files in the `OSGI-INF/blueprint` directory. You can also name specific files using the `Bundle-Blueprint` property in the bundle manifest file. For example:`Bundle-Blueprint: OSGI-INF/blueprint/blueprint.xml,OSGI-INF/blueprint/helloWorldRef.xml`

The following code is an example of the contents of a blueprint file, `OSGI-INF/blueprint/blueprint.xml`:

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">
  <bean activation="lazy" id="helloEBA" class="com.ibm.ws.eba.helloWorld.HelloWorldEBAImpl" init-method="hello">
  </bean>
  <service ranking="0" auto-export="disabled" interface="com.ibm.ws.eba.helloWorld.HelloWorldEBA" ref="helloEBA">
  </service>
</blueprint>
```

For more information about the Blueprint Container specification, see the Compendium Specification on the [OSGi Alliance website](#).

Parent topic: [OSGi Blueprint Container Specification](#)

Parent topic: [Creating blueprint XML files](#)

OSGi architecture

The core part of the OSGi Service Platform defines a secure and managed service platform that is based on Java™. This platform supports the deployment of extensible and downloadable applications that are known as bundles. The specification defines a security model, an application lifecycle management model, a service registry, an Execution environment, and Modules.

OSGi defines a dynamic module system for Java. The core OSGi Service Platform has a layered architecture, and is designed to run on various standard Java profiles. OSGi introduces the notion of a bundle as a modular unit, and the platform architecture is based on bundles as the unit of deployment. The OSGi architecture has the following layers:

- [Execution environment layer](#)
- [Module layer](#)
- [Lifecycle layer](#)
- [Service registry layer](#)

For more information about OSGi architecture, see the [core OSGi service platform specification](#).

Parent topic: [OSGi overview](#)

Execution environment layer

The execution environment layer specifies the Java environment (for example, Java EE or Java SE) under which a bundle runs. For OSGi applications that run in WebSphere® Application Server, you do not need to specify the execution environment.

Module layer

The module layer is where the OSGi Framework processes the modular aspects of a bundle. The metadata that enables the OSGi Framework to process the bundle is set in a bundle manifest file.

One of the key advantages of OSGi is its class loader model, which uses the metadata in the manifest file. OSGi does not have a global class path. When bundles are installed into the OSGi Framework, the metadata is processed by the module layer and the declared external dependencies are reconciled against the versioned exports declared by other installed modules. The OSGi Framework determines the dependencies by using the manifest, and calculates the independent required class path for each bundle. This approach resolves the shortcomings of plain Java class loading by ensuring that the following requirements are met:

- Only packages that are explicitly exported by a particular bundle, through the metadata, are visible to other bundles for import.
- Each package can be resolved to specific versions.
- Multiple versions of a package can be available concurrently to different clients.

Lifecycle layer

The bundle lifecycle management layer in OSGi eliminates the problem with Java class loading and the `class not found` exception at run time, in which dependant classes cannot be loaded because they cannot be found. When an installed bundle is deployed into the framework, the framework first resolves all of its declared dependences. If there are unresolved dependences, the framework reports these dependences and does not start the bundle.

In the bundle lifecycle:

- Bundles are dynamic, and can be started and stopped independent of the rest of the framework.

- Each bundle can provide a bundle activator that the framework calls on start and stop events. The bundle activator is declared in the bundle manifest.

Applications typically do not need to provide a bundle activator. However, if initialization is required when the bundle starts or stops, a bundle activator can be created.

Service registry layer

The service registry layer in OSGi intrinsically supports service-oriented architecture (SOA). Bundles publish services to the service registry, and other bundles can discover these services from the service registry.

These services are the primary means of collaboration between bundles. An OSGi service is a Plain Old Java Object (POJO), published to the service registry under one or more Java interface names, with optional metadata stored as custom properties (name-value pairs). A discovering bundle looks up a service in the service registry by an interface name, and then can potentially filter the services by using the custom properties.

Services are fully dynamic and typically have the same lifecycle as the bundle that provides them.

OSGi bundles

An OSGi bundle is a Java™ archive file that contains Java code, resources, and a manifest that describes the bundle and its dependencies. The bundle is the unit of deployment for an application.

- [Types of bundles](#)
- [OSGi bundle manifest file](#)
- [Bundle lifecycle](#)

Parent topic: [OSGi overview](#)

Parent topic: [Creating OSGi bundle projects](#)

Types of bundles

- Application bundles

- Application bundles are bundles that you create specifically for your application. They are instance-specific or isolated; that is, they are not intended to be shared. They are referenced in the application manifest in the Application-Content header.

- Shared bundles

- Shared bundles are not application-specific. A single instance of a package from a shared bundle can be used by many applications. Shared bundles cannot import packages or services from application bundles. Shared bundles in an application must be provided by reference rather than contained directly in an application.
- Shared bundles are further subdivided into use bundles and provision bundles:

- Use bundles

- A use bundle is a shared bundle that provides at least one package to an application bundle. Use bundles are referenced in the application manifest in the Use-Bundle header.

- Provision bundles

- A provision bundle is a shared bundle that provides at least one package or service to an application bundle, a use bundle, or another provision bundle. Provision bundles are not referenced in the application manifest, and your application does not know how the requirement for each provision bundle is satisfied.

OSGi bundles can be stored in any of the following locations:

- The enterprise bundle archive (EBA) file for the application.
- A server internal OSGi bundle repository.
- External OSGi bundle repositories.

Application bundles can be stored either in the EBA file or in a repository. Shared bundles are stored in a repository (otherwise they cannot be shared).

The process of getting bundles from the repositories is known as provisioning. For provisioning purposes, the following terminology is used for bundles:

- Referenced bundles

- A referenced bundle is a bundle that is referenced in the application manifest, and stored in a repository.

- Dependency bundles

- A dependency bundle is a bundle that is not referenced in the application manifest, but that is used by bundles that are referenced in the application manifest, or by other dependency bundles.

Application bundles that are not directly contained in the EBA file are instance-specific referenced bundles. Use bundles are shared referenced bundles. Provision bundles are shared dependency bundles.

OSGi bundle manifest file

An OSGi bundle JAR file contains a JAR manifest file. This file contains metadata that enables the OSGi Framework to process the modular aspects of the bundle.

The following code is an example of the contents of a bundle manifest file, `META-INF/MANIFEST.MF`: `Manifest-Version: 1.0`

```
Bundle-ManifestVersion: 2
Bundle-Name: MyService bundle
Bundle-SymbolicName: com.sample.myservice
Bundle-Version: 1.0.0
Bundle-Activator: com.sample.myservice.Activator
Import-Package: org.apache.commons.logging;version="1.0.4"
Export-EJB: ExampleBean
Export-Package: com.sample.myservice.api;version="1.0.0"
```

The metadata in this manifest file includes the following key properties:

- Bundle-Version

- Describes the version of the bundle and enables multiple versions of a bundle to be active concurrently in the same framework instance.

- Bundle-Name

- Provides a human readable name for a bundle.

- Bundle-SymbolicName

- Uniquely identifies a bundle in the framework. It does not replace the need for a Bundle-Name header.

- Bundle-Activator

- Receives notification from the framework about the bundle lifecycle changes. This property specifies the class that implements the `org.osgi.framework.BundleActivator` interface.

- Import-Package

- Declares the external dependencies of the bundle that are used by the OSGi Framework for bundle resolution. Specific versions or version ranges for each package can be declared. In this example manifest file, the `org.apache.commons.logging` package is required at Version 1.0.4 or later.
- Use this property to specify the names of any packages that you want your bundle to import from the run time. If you do not specify the package that your bundle needs in this property, you might get a compilation error when the bundle loads. **Note:** You must also specify this package in the Export-Package property of the bundle that contains the package.

Restriction: If your bundle manifest file uses the `Import-Package` property to declare a bundle dependency for a bundle that is not in your workspace or your target platform, the editor marks the dependency as an error. To work around this limitation, make sure that all the bundles that you declare as a dependency are in your workspace, or use the quick fix to add the bundle to the target platform. To use the quick fix, switch to the Markers view and then right-click the error marker and select **Quick Fix**.

Important: When you specify bundle dependencies in your `MANIFEST.MF` file, use the property `Import-Package` instead of `Require-bundle`. If you use `Require-bundle` to specify bundle dependencies, your application does not deploy. `Import-Package` is a more flexible way to declare dependencies:

- You can declare dependencies on the functionality that you need rather than on the bundle where the functionality originated, as it does not add unnecessary dependencies on packages that are not required by your bundle but are included in the dependent bundle.

- You can specify versions or version ranges for the declared package, therefore you do not have to react to changes in different versions of the bundle.

- Export-EJB:

- If an EJB project is converted to an OSGi bundle, and Export-EJB section is added to the manifest. Use this section to list EJBs that you want to expose. EJBs can be added to this section by editing manually or by right-clicking a converted EJB project and selecting **OSGi > Manage EJB exports**.

- Export-Package

- Declares the packages that are visible outside the bundle. Any package not declared here has visibility only within the bundle.
- Use this property to specify the name of any package that you want your bundle to export to the run time. If you do not specify the packages that are required by other bundles in this property, the dependent bundles might not resolve.

Bundle lifecycle

The framework manages the lifecycle of bundles. As you install and run a bundle, it goes through various states.

The possible states of a bundle are:

- INSTALLED

- The bundle is installed, but not all of the dependencies of the bundle have been met. The bundle requires packages that have not been exported by any currently installed bundle.

- RESOLVED

- The bundle is installed and the dependencies of the bundle have been met, but it is not running. If a bundle is started and all of the dependencies of the bundle are met, the bundle skips this state.

- STARTING

- A temporary state that the bundle goes through while the bundle is starting.

- ACTIVE

- The bundle is running.

- STOPPING

- A temporary state that the bundle goes through while the bundle is stopping.

- UNINSTALLED

- The bundle no longer exists in the framework.

OSGi applications

An OSGi application groups a set of bundles to provide a coherent business logic. The application can consist of different bundle types such as web enabled bundles and persistence enabled bundles.

Bundles in one OSGi application are isolated from bundles, services, or packages that are defined in another OSGi application, unless the bundles, services, or packages are explicitly shared by both applications. An OSGi application can either directly contain OSGi bundles or reference bundles that are hosted in an OSGi bundle repository.

The OSGi bundles in the application can share services with other OSGi applications. If an OSGi application produces any external services and references, they are explicitly exposed by declaring them in an application manifest. Similarly, any external services and references that the OSGi application consumes are declared in the application manifest. An OSGi application can also use metadata to enable the sharing of some of its constituent bundles in the system. Sharing in this way can reduce the memory and resource requirements of a system.

An OSGi application contains:

- [Archive content](#)
- [Application manifest](#)
- [Deployment manifest](#)

Parent topic: [OSGi overview](#)

Archive content

The archive content is used in two different ways, depending on whether an Application-Content header is defined in the application manifest.

- If an Application-Content header is not defined, the archive content defines the OSGi application content.
- If an Application-Content header is defined, the archive content defines an initial bundle repository from which bundles can be provisioned. If a bundle with a specific version is installed into a governed repository, and the bundle is also contained in an application archive, the bundle from the archive is used.

Application manifest

The application manifest, `META-INF/APPLICATION.MF`, describes modularity at the application level. By default, when application manifest is not declared, the application content is the set of OSGi bundles contained in the OSGi application and no external services or references are produced or consumed.

An application manifest can contain the following headers:

- **Application-SymbolicName**

- The unique symbolic name of the OSGi application, using similar package notation to Java.

- **Application-Version**

- The version of the application, using OSGi syntax for a bundle version.

- **Application-Name**

- The name of the application.

- **Application-ImportService**

- Declares the external dependencies of the bundle that are used by the OSGi Framework for bundle resolution.

Specific versions or version ranges for each service can be declared.

A set of filters for external services that the application consumes. The application manifest must contain the classes that services require. If this header is not specified, none of the required services are imported.

Specify the required services in a comma-separated list with the service interface name first followed by attributes or directives. `<service identifier>;<directives>;<attributes>`

For example: `test.it;filter="some_filter"`

The Application-ImportService header has the following attribute:

- filter

- An OSGi service filter.

- Application-ExportService

- Declares the services that are visible outside the bundle. Any service not declared here has visibility only within the bundle.

A set of filters for external services that the application produces. If this header is not specified, none of the required services are exported.

Specify the exported services in a comma-separated list with the service interface name first followed by attributes or directives. `<service identifier>;<directives>;<attributes>`

For example: `test.it;filter="some_filter"`

The Application-ExportService header has the following attribute:

- filter

- An OSGi service filter.

- Use-Bundle

- A shared bundle that provides at least one package to an application bundle.

A list of bundles or composite bundles to use to satisfy the package dependencies of bundles in the Application-Content list. Each bundle or composite bundle in the Use-Bundle list must provide at least one package to at least one bundle in the Application-Content list. These bundles are provisioned into the shared bundle space at run time. Often, you do not require a Use-Bundle header, but there are some situations where it is useful. You can use it to restrict the level at which sharing is possible. For example, you can ensure that an application uses the same bundle for package imports that it was tested with. Alternatively, you can ensure that two applications use the same bundle for package imports. By setting the restriction at application level, the bundle can remain flexible.

- Application-WebModules

- List of non-OSGi dynamic web projects that are included in the application.

This header is not part of the OSGi standard.

- Application-Content

- A list of composite bundles, bundle fragments, and bundles with the acceptable range of OSGi version specifications that are included in the application. **Tip:** When including a bundle fragment in the Application-Content list, ensure that you include the host bundle for the fragment.

The format is a comma-separated list of module declarations, where each module declaration uses the following format: `<module identifier>;<directives>;<attributes>`

Typically, the module identifier is the symbolic name of a bundle. To reference a resource that is not a bundle, the module identifier is the path relative to the OSGi application root.

The Application-Content header has the following attribute:

- version

- The version of the module is specified using OSGi syntax for a version range. Specify the minimum version of the application followed by the maximum version to which the application can be upgraded. For example, `"[1.0.0,2.0.0)"` means version 1.0.0 and all later versions up to, but excluding, version 2.0.0.

The Application-Content header defines the important applications that compose the business services, but it does

not define the full list of bundles in the application. If a bundle that is listed in the content uses a package that is not included in the application, a dependency analysis is performed and any missing bundles are included. These bundles cannot provide services external to the application and cannot have security applied to them. Bundles that are included using this mechanism are shared.

Deployment manifest

When an OSGi application is installed, the application manifest specifies the bundles that comprise an application. It can specify more than one version for some bundles.

The deployment manifest, `META-INF/DEPLOYMENT.MF`, specifies all the bundles that make up the application, including bundles that are required following dependency analysis. The deployment manifest specifies the actual version of each bundle that is used in the application. It is created automatically when an Enterprise Bundle Archive (EBA) asset is installed and it ensures that each time an application server starts, the bundles that make up the application are the same.

An EBA defines a set of OSGi bundles that are deployed as a single OSGi application, and that are isolated from other OSGi applications. An EBA file is a single archive file with a `.eba` file extension. It contains either a set of application modules or an application manifest, or both.

After an application is installed, the version of a bundle can be updated by configuring the EBA asset.

A deployment manifest contains the following headers:

- **Application-SymbolicName**

- The unique symbolic name of the application, using similar package notation to Java. This matches the Application-SymbolicName value in the application manifest.

- **Application-Version**

- The version of the application, using OSGi syntax for a bundle version. This matches the Application-Version value in the application manifest.

- **Deployed-Content**

- A comma-separated list of the symbolic names of the bundles and the exact versions to use.

The list includes all the bundles that are listed in the Application-Content header in the application manifest, and bundles that are imported by dependency analysis. Non-OSGi module types are included using the symbolic name of the converted bundle.

The Deployed-Content header has the following directive:

- **deployed-version**

- The exact version of the bundle, specified using OSGi syntax for a version.

OSGi fragments

An OSGi fragment is a Java™ archive file with specific manifest headers that enable it to attach to a specified host bundle or specified host bundles to function. Fragments are treated as part of the host bundles. Relevant definitions of the fragment are merged with the host bundles definitions before the host is resolved, provided the information does not conflict. Fragment dependencies are resolved if possible. If the fragment dependencies cannot be resolved, the fragment does not attach to the host bundle. A fragment cannot have its own class loader or bundle activator. It cannot override the information present in the host bundles. Fragments extend bundles with resources, classes, and permitted headers, which enable you to customize your bundles.

OSGi fragment manifest files

An OSGi fragment JAR file contains a fragment manifest file. This file contains metadata that enables the OSGi Framework to attach the fragment to a host bundle or host bundles.

The following code is an example of the contents of a bundle manifest file, `META-INF/MANIFEST.MF`: `Manifest-Version: 1.0`

```
Bundle-ManifestVersion: 2
Bundle-Name: Fragment
Bundle-SymbolicName: Fragment
Bundle-Version: 1.0.0.qualifier
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Fragment-Host: WebBundle;bundle-version=1.0.0.qualifier
Import-Package: org.apache.commons.logging;version="1.0.4"
Export-Package: com.sample.myservice.api;version="1.0.0"
```

The metadata in this manifest file includes the following key properties:

- Fragment-Host

- Links the fragment to its potential bundle hosts.

- Bundle-Version

- Describes the version of the fragment and enables multiple versions of a bundle to be active concurrently in the same framework instance.

- Bundle-Name

- Provides a human readable name for the fragment.

- Bundle-SymbolicName

- Uniquely identifies the fragment in the framework. It does not replace the need for a Bundle-Name header.

- Import-Package

- Declares the external dependencies of the fragment that are used by the OSGi Framework for fragment resolution. Specific versions or version ranges for each package can be declared. In this example manifest file, the `org.apache.commons.logging` package is required at Version 1.0.4 or later.
- Use this property to specify the names of any packages that you want your fragment to import from the run time. If you do not specify the package that your bundle needs in this property, you might get a `NoClassDefFound` exception and a compilation error when the bundle loads. **Note:** You must also specify this package in the `Export-Package` property of the bundle that contains the package.

Restriction: If your bundle manifest file uses the `Import-Package` property to declare a bundle dependency for a bundle that is not in your workspace or your target platform, the editor marks the dependency as an error. To work around this limitation, make sure that all the bundles that you declare as a dependency are in your workspace, or use the quick fix to add the bundle to the target platform. To use the quick fix, switch to the Markers view and then right-click the error marker and select **Quick Fix**.

Important: When you specify bundle dependencies in your `MANIFEST.MF` file, use the property `Import-Package` instead of `Require-Bundle`. If you use `Require-Bundle` to specify bundle dependencies, your application does not deploy. `Import-Package` is a more flexible way to declare dependencies:

- You can declare dependencies on the functionality that you need rather than on the bundle where the functionality originated, as it does not add unnecessary dependencies on packages that are not required by your bundle but are included in the dependent bundle.
- You can specify versions or version ranges for the declared package, therefore you do not have to react to changes in different versions of the bundle.

- Export-Package

- Declares the packages that are visible outside the fragment. Any package not declared here has visibility only within the fragment.
- Use this property to specify the name of any package that you want your fragment to export to the run time. If you do not specify the packages that are required by other bundles in this property, the dependent bundles might not resolve.

Parent topic:[OSGi overview](#)

Parent topic:[Creating fragment projects](#)

OSGi composite bundles

Composite bundles group bundles into aggregates to ensure consistent behavior. A composite bundle contains bundles or references to bundles outside of the workspace or target platform. A composite bundle ensures consistent behavior from a set of shared bundles at a specific version.

Important: Applicable edition: Full profile

A composite bundle archive (CBA) groups shared bundles together into aggregates. A CBA can contain OSGi bundles or reference bundles that are hosted in the internal bundle repository. Create a CBA when you want to ensure consistent behavior across a set of shared bundles. You can use the CBA to wire that set of bundles, at a specific version, to an application.

A CBA is an archive file with a `.cba` file extension. It contains a composite manifest `META-INF/COMPOSITEBUNDLE.MF`, which defines the CBA, and optionally some OSGi bundles with which to seed the repository. The bundles that a CBA contains or references are defined with exact versions, in contrast to an EBA, where bundles are defined with version ranges.

A composite bundle is installed in the internal bundle repository of the run time. If the CBA directly contains OSGi bundles, these bundles are installed into the repository as though they were individually uploaded. The CBA is also added to the bundle repository. If the CBA references OSGi bundles, these bundles must be present in the internal bundle repository. After a CBA is installed in the internal bundle repository, its bundles are available to all applications that want to use the bundles when the application is resolved. If a required package or service is available at the same version from both a bundle and a CBA, the provisioning process selects the package or service from the CBA.

A CBA has the following differences from an enterprise bundle archive (EBA) file:

- A CBA has a composite manifest, which is a modularity statement that asserts that bundles can be deployed, not that they will resolve. An EBA file has an application manifest, which is a provisioning statement.
- A CBA can import or export packages, but an EBA file cannot.
- An EBA file does not need to fully define its content. When the application is deployed, a dependency analysis is performed and additional bundles can be provisioned.
- An EBA file can define that bundles in its content are shared.
- Bundles in a CBA are visible in the repository, but bundles in an EBA file are private to the application.

Parent topic: [OSGi overview](#)

Parent topic: [Creating composite bundle projects](#)

OSGi composite bundle manifest file

The metadata in this file includes the following key properties:

- **Manifest-Version**

- Describes the version of the manifest file.

- **CompositeBundle-ManifestVersion**

- Describes the version of the composite bundle.

- **Bundle-Name**

- Provides a human readable name for a composite bundle.
- If you do not specify a value, the default value is the composite bundle symbolic name.

- **Bundle-SymbolicName**

- Uniquely identifies a composite bundle in the framework. It does not replace the need for a Bundle-Name header.

- **Bundle-Version**

- Describes the version of the composite bundle and enables multiple versions of a composite bundle to be active concurrently in the same framework instance.

- **CompositeBundle-Content**

- A list of bundles and bundle fragments in the composite bundle. All bundles and fragments must be available for deployment and must be contained in the `cba` file, or exist in the local bundle repository. Bundles and fragments must have exact version numbers. If you require the same composite bundle with different versions of its content, you need to create different versions of the composite bundle, one version for each use.
- The format is a comma-separated list of module declarations, where each module declaration uses the following format: `<module identifier>;<directives>;<attributes>`
- The `CompositeBundle-Content` header has the following attribute:
 - **version**
 - The version of the module is specified using OSGi syntax for a version range. The version must be specified as an exact range, for example `"[1.0.0,1.0.0]"`.

- **Import-Package**

- Declares the external dependencies of the bundle that are used by the OSGi Framework for bundle resolution. Specific versions or version ranges for each package can be declared.
- Use this property to specify the names of any packages that you want your bundle to import from the run time. If you do not specify the package your bundle needs in this property, you might get a `NoClassDefFound` exception when the bundle loads.
- **Note:** You must also specify the package that you want to import (by using `Import-Package`) in the `Export-Package` property of the bundle that contains the package.

- **Export-Package**

- Declares the packages that are visible outside the bundle. Any package not declared here has visibility only within the bundle.
- Use this property to specify the name of any package that you want your bundle to export to the run time. If you do not specify the packages that are required by other bundles in this property, the dependent bundles might not resolve.

- **CompositeBundle-ExportService**

- A list of service interface names and optional filters that identify services that are present in the composite bundle and that can be exported for use outside the composite bundle. The interfaces that an exported service implements are usable outside the composite bundle if those interfaces are visible outside the composite bundle. The format is a comma-separated list of services, in the form of a service interface name, followed by attributes or directives. The `CompositeBundle-ExportService` header has the following attribute:
 - **filter**
 - An OSGi service filter.

- **CompositeBundle-ImportService**

- A list of service interface names and optional filters that identify services that the contents of the composite bundle want to use from outside the composite bundle. At least one such service must exist at run time. The format is a comma-separated list of services, in the form of a service interface name, followed by attributes or directives. The `CompositeBundle-ImportService` header has the following attribute:
 - **filter**
 - An OSGi service filter.

Tools for OSGi application development

The OSGi application development tools help you to create OSGi applications.

The development environment contains a selection of views, wizards, and editors that are customized to be the most useful for an OSGi developer.

The following list of editors range from standard source editors with content assist features to more full featured editors:

- **Bundle manifest editor** helps you to describe your bundle and bundle dependencies. For more information, see [OSGi bundle manifest file](#).
- **OSGi composite bundle manifest editor** helps you describe your composite bundle and bundle dependencies. For more information, see [Editing composite bundle manifest files](#).
- **Blueprint XML editor** helps you to define and describe the various components of an application. For more information, see [OSGi Blueprint XML files](#).
- **OSGi application manifest editor** helps you define and describe the metadata that enables the framework to process the modular aspects of your bundles. For more information, see [Application manifest files](#).
- **OSGi fragment manifest editor** helps you define and describe your bundle fragment and host bundle. For more information, see [Fragment manifest files](#).
- **OSGi Bundle Explorer** visualizes your bundles and the dependencies between them. For more information, see [Exploring bundle dependencies](#).

Parent topic: [Developing OSGi applications](#)

Installing the server

About this task

Before you can begin developing OSGi applications, you must install one of the following products:

- The IBM® WebSphere® Application Server Version 7.0 Feature Pack for OSGi Applications and Java™ Persistence API 2.0
- IBM WebSphere Application Server Version 8.0
- IBM WebSphere Application Server Version 8.5
- IBM WebSphere Application Server Liberty profile.

You can develop the following Java EE applications as OSGi applications:

- Web applications
- Java Persistence API (JPA) applications
- JAX-RS (REST) Web services
- JavaServer Faces applications
- XML applications
- EJBs

Table 1. WebSphere Application Server support for OSGi applications

	Version support in WebSphere Application Server version 7.0	Version support in WebSphere Application Server version 8.0	Version support in WebSphere Application Server version 8.5	Version support in Liberty profile server
Dynamic Web Modules	2.5	2.5 and 3.0	2.5 and 3.0	2.5 and 3.0
EJBs	Not supported	Not supported	3.0 and 3.1	Not supported
Java Persistence API	1.0 and 2.0	1.0 and 2.0	1.0 and 2.0	1.0 and 2.0
JavaServer Faces	1.1 and 1.2	1.1, 1.2, and 2.0	1.1, 1.2, and 2.0	Not supported
JAX-RS	1.1	1.1	1.1	1.1
SIP	1.0 and 1.1 Note: Annotated SIP1.1, or OSGi, applications are not supported	1.0 and 1.1 Note: Annotated SIP1.1, or OSGi, applications are not supported	1.0 and 1.1 Note: Annotated SIP1.1, or OSGi, applications are not supported	Not supported

Restriction:The following technologies are not supported by the WebSphere Application Server Version 7.0 OSGi run time:

- EJB
- RPC Adapter
- Servlet 3.0
- Web 2.0 Server-side technologies
- Web 3.0
- Web services
- XML Transformations APIs

The following technologies are not supported by the WebSphere Application Server Version 8.0 OSGi run time:

- EJB
- RPC Adapter
- Web 2.0 Server-side technologies
- Web services
- XML Transformations APIs

For more information about OSGi applications support in WebSphere Application Server, see the [WebSphere Application Server documentation](#).

Parent topic: [Developing OSGi applications](#)

Installing WebSphere Application Server Version 7.0 Feature Pack for OSGi Applications and Java Persistence API 2.0 Procedure

1. Open the IBM Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 7.0 Test Environment**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 7.0.
6. In the Features list, ensure that you select **OSGi Applications** under **IBM WebSphere Application Server Version 7.0 Feature Pack for OSGi Applications and Java Persistence API 2.0**.

Installing WebSphere Application Server Version 8.0 Procedure

1. Open the IBM Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 8.0**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.0.

Installing WebSphere Application Server Version 8.5 Procedure

1. Open the IBM Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server Version 8.5**, then click **Next**.
4. Read the license agreements. Accept the license agreements then click **Next**.
5. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.5.

Installing WebSphere Application Server Liberty profile

About this task

For details about installing the Liberty profile, see [Installing Liberty](#).

Creating OSGi projects

OSGi projects hold all of the resources that you create, maintain, and use as you develop your applications.

About this task

You can develop the following Java™ EE applications as OSGi applications:

- Web applications
- Java Persistence API (JPA) applications
- JAX-RS (REST) Web services
- JavaServer Faces applications
- XML applications
- EJBs

Procedure

1. Read about OSGi project facets.
2. Create and configure an OSGi project.

- **OSGi project facets**

A project facet is a specific unit of functionality that you can add to a project when that functionality is required. When a project facet is added to a project, it can add natures, builders, class path entries, and resources to a project, depending on the characteristics of the particular project. Facets define characteristics and requirements for OSGi projects and are used as part of the runtime configuration.

- **Creating OSGi bundle projects**

An OSGi bundle is a Java archive file that contains Java code, resources, and a manifest that describes the bundle and bundle dependencies. An OSGi bundle contains the business logic and metadata that you need to run a service. A bundle is a module in an application, which in turn is deployed to a server.

- **Creating blueprint XML files**

The blueprint configuration file contains the component assembly and configuration information for a bundle. The file describes how components are registered in the OSGi service registry or how they look up services from the OSGi service registry. This information is used at run time to instantiate and configure the required components when the bundle is started.

- **Creating fragment projects**

Fragments extend bundles with resources, classes, and permitted headers, which you can use to customize your bundles.

- **Creating composite bundle projects**

Composite bundles group bundles into aggregates to ensure consistent behavior. A composite bundle contains bundles or references to bundles outside of the workspace or target platform. A composite bundle ensures consistent behavior from a set of shared bundles at a specific version.

- **Creating OSGi application projects**

An OSGi application project groups a set of bundles to provide a coherent business logic. The application can consist of different bundle types such as web enabled bundles and persistence (JPA) enabled bundles.

Parent topic: [Developing OSGi applications](#)

OSGi project facets

A project facet is a specific unit of functionality that you can add to a project when that functionality is required. When a project facet is added to a project, it can add natures, builders, class path entries, and resources to a project, depending on the characteristics of the particular project. Facets define characteristics and requirements for OSGi projects and are used as part of the runtime configuration.

When you add a facet to a project, that project is configured to perform a certain task, fulfill certain requirements, or have certain characteristics. For example, the EAR facet sets up a project to function as an enterprise application by adding a deployment descriptor and setting up the project class path.

Some facets require other facets as prerequisites. In other cases, facets cannot coexist in the same project. For example, you cannot add the Dynamic Web Module facet to an EJB project because the EJB project already has the EJB Module facet. Some facets can be removed from a project and others cannot.

Table 1. OSGi application development facets

Project facet	Description	Dependencies
OSGi Application	This facet supplies the basic behaviors and capabilities that are associated with OSGi applications, such as:Extensions in Enterprise Explorer that visualize the contents of the applicationAPPLICATION.MF editingCapability to export to an EBA file	
OSGi Bundle	This facet supplies the basic behaviors and capabilities that are associated with OSGi bundles, such as:OSGi blueprint service indexingCapability to export to a bundle JAR file	Java™ version 5.0+
OSGi Composite Bundle	This facet supplies the basic behaviors and capabilities that are associated with OSGi composite bundles, such as:Extensions in Enterprise Explorer that visualize the contents of a composite bundleCOMPOSITEBUNDLE.MF editingCapability to export to a CBA file	

OSGi Fragment	This facet supplies the basic behaviors and capabilities that are associated with OSGi fragments, such as:Capability to extend an OSGi bundleCapability to export a fragment JAR file	Java version 1.5+
---------------	---	-------------------

Parent topic:[Creating OSGi projects](#)

Creating OSGi bundle projects

An OSGi bundle is a Java™ archive file that contains Java code, resources, and a manifest that describes the bundle and bundle dependencies. An OSGi bundle contains the business logic and metadata that you need to run a service. A bundle is a module in an application, which in turn is deployed to a server.

Procedure

1. Click **File > New > Other > OSGi > OSGi Bundle Project** and then click **Next**. The New OSGi Bundle Project wizard opens.
2. In the **Project name** field, enter the name of your bundle project.
3. Select a **Target runtime** from the drop-down list. Set the target run time to define an installed runtime environment. Run times are used at build time to compile projects
4. In the Configuration list, you can select one or more of the following configurations:
 - **Add Web support** - Adds support for dynamic web page content to your bundle project.
 - **Add persistence support** - Adds JPA support to your bundle project.
 - **Custom** - Adds support to deploy your project as an OSGi bundle project so that you can add additional facets to your project. Click **Advanced** - Adds facets to your project. **Important:** You can develop the following Java EE applications as OSGi applications:
 - Web applications
 - Java Persistence API (JPA) applications
 - JAX-RS (REST) Web services
 - JavaServer Faces applications
 - XML applications
 - EJBs
 - **Generate blueprint file** - Creates a blueprint file and adds it to your bundle project.
5. In the Application membership section, select **Add bundle to application** and then select an application project from the drop-down list or create a new application project.
6. Follow the wizard prompts. **Restriction:** If your bundle manifest file uses the `Import-Package` property to declare a bundle dependency for a bundle that is not in your workspace or your target platform, the editor marks the dependency as an error. To work around this limitation, make sure that all the bundles that you declare as a dependency are in your workspace, or use the quick fix to add the bundle to the target platform. To use the quick fix, switch to the Markers view and then right-click the error marker and select **Quick Fix**.

Results

Your OSGi bundle project is created and your bundle manifest is added to your project. **Important:** When you specify bundle dependencies in your `MANIFEST.MF` file, use the property `Import-Package` instead of `Require-bundle`. If you use `Require-bundle` to specify bundle dependencies, your application does not deploy. `Import-Package` is a more flexible way to declare dependencies:

- You can declare dependencies on the functionality that you need rather than on the bundle where the functionality originated, as it does not add unnecessary dependencies on packages that are not required by your bundle but are included in the dependent bundle.
- You can specify versions or version ranges for the declared package, therefore you do not have to react to changes in different versions of the bundle.

What to do next

Now that you have created a bundle project, you can create your business logic, export the OSGi bundle as a JAR file, add the bundle to an OSGi application, or add the bundle to a composite bundle.

- **OSGi bundles**

An OSGi bundle is a Java archive file that contains Java code, resources, and a manifest that describes the bundle and its dependencies. The bundle is the unit of deployment for an application.

- **Creating web enabled OSGi bundle projects**
- **Creating JPA enabled OSGi bundle projects**
- **Creating JAX-RS enabled OSGi bundle projects**
- **Editing bundle manifest files**

Parent topic: [Creating OSGi projects](#)

Creating web enabled OSGi bundle projects

Procedure

1. Click **File > New > Other > OSGi > OSGi Bundle Project** and then click **Next**. The New OSGi Bundle Project wizard opens.
2. In the **Project name** field, enter the name of your bundle project.
3. Select a **Target runtime** from the drop-down list. Set the target run time to define an installed runtime environment. Run times are used at build time to compile projects
4. In the Configuration list, select **Add Web support**. This configuration adds support for dynamic web page content to your bundle project.
5. Select **Generate blueprint file** to generate a blueprint file for your bundle project.
6. In the Application membership section, select **Add bundle to application** and then select an application project from the drop-down list or create a new application project.
7. Follow the wizard prompts.

Parent topic: [Creating OSGi bundle projects](#)

Creating JPA enabled OSGi bundle projects

Procedure

1. Click **File > New > Other > OSGi > OSGi Bundle Project** and then click **Next**. The New OSGi Bundle Project wizard opens.
2. In the **Project name** field, enter the name of your bundle project.
3. Select a **Target runtime** from the drop-down list. Set the target run time to define an installed runtime environment. Run times are used at build time to compile projects
4. In the Configuration list, select **Add persistence support**.
5. Select **Generate blueprint file** to generate a blueprint file for your bundle project.
6. In the Application membership section, select **Add bundle to application** and then select an application project from the drop-down list or create a new application project.
7. Follow the wizard prompts.

What to do next

After you create the JPA enabled OSGi bundle project, you can access data using JPA technology.

Parent topic: [Creating OSGi bundle projects](#)

Creating JAX-RS enabled OSGi bundle projects

Procedure

1. Click **File > New > Other > OSGi > OSGi Bundle Project** and then click **Next**. The New OSGi Bundle Project wizard opens.
2. In the **Project name** field, enter the name of your bundle project.
3. Select a **Target runtime** from the drop-down list. Set the target run time to define an installed runtime environment. Run times are used at build time to compile projects
4. In the Configuration list, select **Custom** and then click **Advanced**. The Project Facets dialog opens.
5. Select **JAX-RS (REST Web Services)** and then click **OK**.
6. Select **Generate blueprint file** to generate a blueprint file for your bundle project.
7. In the Application membership section, select **Add bundle to application** and then select an application project from the drop-down list or create a new application project.
8. Follow the wizard prompts.

Parent topic: [Creating OSGi bundle projects](#)

Editing bundle manifest files

About this task

The OSGi bundle manifest file contains metadata that enables the OSGi Framework to process the modular aspects of the bundle.

Procedure

1. In Enterprise Explorer, expand your bundle project.
2. Double-click **Manifest:<name>**, where *<name>* is the name of your bundle, to open your bundle manifest in the editor.

What to do next

For more information about the OSGi bundle manifest file, see [OSGi bundle manifest file](#), and the following subtopics:

- [Declaring dependencies outside of the workspace](#)
- [Declaring dependencies outside of the workspace that resolve at run time](#)
- [Specifying version range](#)
- [Exploring bundle dependencies](#)

The bundle explorer visually represents your bundles and bundle dependencies.

Parent topic: [Creating OSGi bundle projects](#)

Declaring dependencies outside of the workspace

About this task

The bundle manifest file tracks all dependent projects, JAR files, and classes. To declare a dependency outside of the workspace, you need to add a `Bundle-ClassPath` entry to your bundle manifest file.

Procedure

1. Right-click the bundle project for which you want to declare an external dependency and select **Properties**. The Properties window opens.
2. From the properties list, select **Deployment Assembly**. The OSGi Bundle Module Assembly properties page opens.
3. Click **Add** and then select the type of reference that you want to add to the bundle.
4. Follow the instructions in the wizard.

Results

The `Bundle-ClassPath` entry is added to your bundle manifest file and when you deploy the bundle on the server, the external dependencies are automatically exported to the server.

Parent topic: [Editing bundle manifest files](#)

Declaring dependencies outside of the workspace that resolve at run time

About this task

When you declare a dependency for a bundle that is outside of the workspace, you receive an error marker in the source view of your bundle manifest file at the `Import-Package` header. To resolve this dependency at run time, you need to add it to your target platform or to your workspace

Procedure

1. Switch to the Markers view. Click **Window > Show View > Markers**.
2. Right-click the error and select **Quick Fix**. The Quick Fix dialog opens.
3. Select one of the following options:
 - Import a bundle to the workspace.
 - Add a bundle to the target platform.
4. Click **Finish**.
5. A wizard opens to assist you with importing the bundle or adding the bundle to the target platform. Follow the instructions in the wizard to import the bundle into your workspace or add it to your target platform.

Parent topic: [Editing bundle manifest files](#)

Specifying version range

About this task

The version of the bundle or package is specified by using OSGi syntax for a version range. The version range is specified by using interval notation.

- The open parentheses () are used to denote that the value is not included in the range. The value is exclusive.
- The closed parentheses [] are used to denote that the value is included in the range. The value is inclusive.

For example, the bundle with the version range `version="[1.0.0,2.0.0)"` includes version 1.0.0 but excludes version 2.0.0.

- [Specifying package and bundle version ranges in a bundle](#)
- [Specifying bundle, composite, and fragment version ranges in an application](#)
- [Specifying the bundle and fragment version in a composite bundle](#)
- [Specifying host bundle version ranges in a fragment bundle](#)

Parent topic: [Editing bundle manifest files](#)

Specifying package and bundle version ranges in a bundle

Procedure

1. Open the bundle manifest file in the editor.
2. Switch to the Dependencies tab. In the Imported Packages section, select the package for which you want to specify version information. Click **Properties**.
3. In the dialog, specify the minimum and maximum versions and if the values are inclusive or exclusive.
4. Save the file.

Specifying bundle, composite, and fragment version ranges in an application

Procedure

1. Open the application manifest file in the editor.
2. In the Contained Bundles section, select the bundle for which you want to specify version information. Click **Properties**.
3. In the OSGi bundle properties dialog, specify the minimum and maximum versions and if the values are inclusive or exclusive.
4. Save the file.

Specifying the bundle and fragment version in a composite bundle

Procedure

1. Open the composite manifest file in the editor.

2. In the Contained Bundles section, select the bundle for which you want to specify version information. Click **Properties**.
3. In the OSGi bundle properties dialog, specify the version. Bundles and fragments must have exact version numbers. If you require the same composite bundle with different versions of its content, you must create different versions of the composite bundle, one version for each use.
4. Save the file.

Specifying host bundle version ranges in a fragment bundle Procedure

1. Open the fragment bundle manifest file in the editor.
2. Switch to the Overview tab. In the General Information section, specify the host bundle and minimum and maximum version information.
3. Save the file.


Exploring bundle dependencies

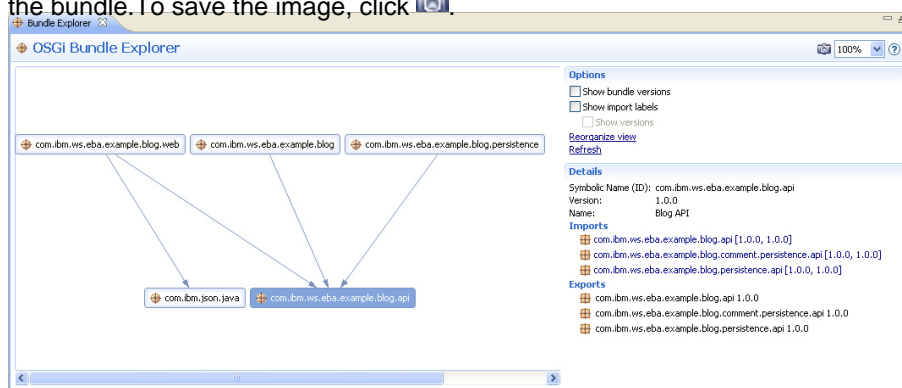
The bundle explorer visually represents your bundles and bundle dependencies.

Procedure

In Enterprise Explorer, right-click your project and then select **Show In > Bundle Explorer**.

Results

The OSGi Bundle Explorer opens in the editor area. In the diagram, select a bundle to display detailed information about the bundle. To save the image, click .



Parent topic: [Editing bundle manifest files](#)

Creating blueprint XML files

The blueprint configuration file contains the component assembly and configuration information for a bundle. The file describes how components are registered in the OSGi service registry or how they look up services from the OSGi service registry. This information is used at run time to instantiate and configure the required components when the bundle is started.

Procedure

1. Right-click your bundle project and select **New > Blueprint File**. The New Blueprint File wizard opens.
2. In the **File name** field, type the name of your blueprint configuration file and then click **Next**.
3. In the Add or Remove Additional Blueprint Namespaces page of the wizard, select the namespaces that you want to add to your blueprint file. You can add namespaces for blueprint extensions, JPA, Resource reference, and Transactions.

Learn more about blueprint namespaces: *Table 1. Description of blueprint namespaces*

Schema	URL	Description
Blueprint extension	http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.0.0	IBM® Blueprint extensions Extensions to the OSGi Blueprint programming model, such as field injection.
Blueprint security	http://www.ibm.com/appserver/schemas/blueprint/security/v1.0.0	IBM Blueprint security Configure bean security so that the methods of the bean can be accessed only by users that are assigned a specified role. More information: Blueprint security and OSGi applications
JPA	http://aries.apache.org/xmlns/jpa/v1.0.0	JPA Blueprint support Provides Java™ Persistence API (JPA) integration into Blueprint. Persistence units and persistence contexts can be injected into Blueprint managed beans.
Resource reference	http://www.ibm.com/appserver/schemas/8.0/blueprint/resourcereference	Blueprint resource reference support Provides for objects that are declared in WebSphere® Application Server JNDI, such as data sources and connection factories to be injected into Blueprint managed beans.

Transactions	http://aries.apache.org/xmlns/transactions/v1.0.0	Blueprint transaction support Allows the developer to declare transaction requirements for Blueprint managed beans.
--------------	---	--

Note: If you want to add other namespaces not in this list, switch to the Source view and type in the namespace.

4. Click **Finish**. The blueprint configuration file opens in the editor.

5. Click **Add** to add the component assembly and configuration information to your blueprint configuration file.

- Bean

- The `bean` element defines the blueprint component that is instantiated. Click **OK** to add arguments and properties to the bean.

- Learn more about Bean attributes:

-

ID

- The `id` attribute identifies the component. It is mandatory if the component is referenced from elsewhere in the blueprint, for example if it is referenced from a service definition.

- Class

- The `class` attribute specifies which implementation class of the component is instantiated.

-

Activation

- This optional attribute defines the activation mode for the manager. Two activation modes are supported:

- eager

- The manager is activated during Blueprint Container initialization.

- lazy

- The manager is activated on demand.

By default activation is set to eager.

- Scope

- Depending on the scope setting, a bean manager can create single or multiple object instances. The Blueprint Container specification defines two main scopes:

- singleton

- The bean manager creates a single instance of the bean and returns that instance every time the manager is asked to provide an object.

- prototype

- The bean manager creates a new instance of the bean every time the manager is asked to provide an object.

By default, the singleton scope is assumed for top-level bean managers. The scope attribute cannot be set on an inlined bean manager, so the inlined managers are always considered to have prototype scope.

-

Depends On

- Specifies a list of manager IDs. The listed managers are activated first before the manager is activated. A manager can have explicit and implicit dependencies. The `dependsOn` attribute defines the explicit dependencies. The implicit dependencies are defined by the references to other managers within a manager definition.

- **Factory reference**

- Specifies the ID of a bean or reference manager that acts as a factory. The specified object must have a factory method as specified by the factory method attribute.

- **Destroy method**

- Specifies a method to be called when the Blueprint Container is destroying the object instance.

- **Factory method**

- Specifies the name of the static factory method.

- **Initialization method**

- The initialization method is called when the component is created. If you do not want to invoke a method during bundle initialization, remove this attribute.

- **Reference**

- The `reference` element specifies services that are found in the service registry. Click **OK** to add items to the reference.

- **Learn more about Reference attributes:**

- **ID**

- The `id` attribute identifies the component. It is mandatory if the component is referenced from elsewhere in the blueprint, for example if it is referenced from a service definition.

- **Interface**

- The `interface` attribute refers to the interface that the component class implements.

- **Activation**

- This optional attribute defines the activation mode for the manager. Two activation modes are supported:

- **eager**

- The manager is activated during Blueprint Container initialization.

- **lazy**

- The manager is activated on demand.

By default activation is set to eager.

-

Availability

- Controls requirements by the service reference manager that at least one service, which matches selection criteria, exists before the blueprint container initialization continues. The availability attribute has two values:

- **optional**

- Services matching the criteria do not have to exist.

- **mandatory**

- At least one service that matches the criteria must exist.

By default availability is set to mandatory.

-

Filter

- Specifies the filter expression for service selection.

- **Timeout**

- Specifies the amount of time, in milliseconds, a proxy waits for a backing service to become available.

-

Component name

- Specifies the component searched by the filter.

- **Depends On**

- Specifies a list of manager IDs. The listed managers are activated first before the manager is activated. A manager can have explicit and implicit dependencies. The `dependsOn` attribute defines the explicit dependencies. The implicit dependencies are defined by the references to other managers within a manager

definition.

- Reference List

- The `reference-list` element enables the reference element to find multiple matching services. Click **OK** to add items to the reference list.

- Learn more about Reference list attributes:

- ID

- The `id` attribute identifies the component. It is mandatory if the component is referenced from elsewhere in the blueprint, for example if it is referenced from a service definition.

- Interface

- The `interface` attribute refers to the interface that the component class implements.

- Activation

- This optional attribute defines the activation mode for the manager. Two activation modes are supported:

- eager

- The manager is activated during Blueprint Container initialization.

- lazy

- The manager is activated on demand.

By default activation is set to eager.

- Availability

- Controls requirements by the service reference manager that at least one service, which matches selection criteria, exists before the blueprint container initialization continues. The availability attribute has two values:

- optional

- Services matching the criteria do not have to exist.

- mandatory

- At least one service that matches the criteria must exist.

By default availability is set to mandatory.

- Filter

- Specifies the filter expression for service selection.

- Member type

- Specifies the type of members provided by the reference list manager. The member type attribute supports:

- service-object

- Injects a list of service proxy objects.

- service-reference

- Injects a list of service reference objects.

By default member type is set to service-object.

- Component name

- Specifies the component searched by the filter.

- Depends On

- Specifies a list of manager IDs. The listed managers are activated first before the manager is activated. A manager can have explicit and implicit dependencies. The `dependsOn` attribute defines the explicit dependencies. The implicit dependencies are defined by the references to other managers within a manager definition.

- Service

- The `service` element defines the export of a component to the OSGi service registry. Click **OK** to add items to the service.

- Learn more about Service attributes:

- ID

- The `id` attribute identifies the component. It is mandatory if the component is referenced from elsewhere in the blueprint, for example if it is referenced from a service definition.

- **Interface**

- The `interface` attribute refers to the interface that the component class implements.

- **Activation**

- This optional attribute defines the activation mode for the manager. Two activation modes are supported:

- **eager**

- The manager is activated during Blueprint Container initialization.

- **lazy**

- The manager is activated on demand.

By default activation is set to eager.

- **Auto-Export**

- The auto-export setting is specified by the `auto-export` attribute and supports the following four options:

- **disabled**

- The default value if the `auto-export` attribute is not specified. The list of interfaces must be specified by using the `interface` attribute or `interfaces` subelements.

- **interfaces**

- Register the service by using all public interfaces implemented by the service class and any of its super classes.

- **class-hierarchy**

- Register the service by using the service class and any of its public super classes.

- **all-classes**

- Combines `interfaces` and `class-hierarchy` options.

- **Ranking**

- You can use the `ranking` attribute to expose the service with a specific ranking.

- **Reference**

- The `ref` attribute refers to the component id of the exported component. This id is defined in the component element.

- **Depends On**

- Specifies a list of manager IDs. The listed managers are activated first before the manager is activated. A manager can have explicit and implicit dependencies. The `dependsOn` attribute defines the explicit dependencies. The implicit dependencies are defined by the references to other managers within a manager definition.

- **Type Converters**

- The `type-converters` element converts values between data types. Click **OK** to add items to the type converter.

For more information about the blueprint configuration file, see the Help topics [OSGi Blueprint XML files](#) and [OSGi Blueprint Container Specification](#).

6. Click **Open WebSphere Blueprint Bindings Descriptor** to create an OSGi blueprint configuration file that contains the resource references to authentication alias binding for a bundle. If an OSGi application contains an OSGi blueprint binding configuration file when it is deployed as an asset, the binding configuration file provides the default authentication alias values that are used when binding the resource references.

For more information about the blueprint binding file, see [Creating blueprint binding XML files](#).

7. Click **Add and remove additional blueprint namespaces** to add namespaces to the blueprint XML file.
 - A. In the Blueprint Namespaces dialog, select the namespaces that you want to add to the blueprint file.
 - B. Click **Finish**.

These elements are enabled based on the blueprint namespaces that you add to the blueprint XML file.

- **Context**

- The `bpjpa:context` element extends the [Apache Aries](#) blueprint specification for JPA beans used within blueprint transactions. This element is added when you add the JPA namespace. Click **Add** to add attributes to the element.

- **Properties placeholder**

- The `bpext:property-placeholder` element adds extensions to the [Apache Aries](#) blueprint specification. This element is added when you add the Blueprint extension namespace. Click **Add** to add attributes to the element.

- **Resource reference**

- The `bpresref:resource-reference` element injects resource references, for example data sources, into beans or services. This element is added when you add the Resource reference namespace. Click **Add** to add attributes to the element.

- **Transaction**

- The `tx:transaction` element extends the [Apache Aries](#) blueprint specification by adding transaction properties to the blueprint file. This element is added when you add the Transaction namespace. Click **Add** to add attributes to the element.

8. Save the file. For more information about OSGi blueprint XML files, see the following subtopics:

- **OSGi Blueprint XML files**

Blueprint XML files define and describe the various components of an application.

- **Creating blueprint binding XML files**

The blueprint binding XML file, `ibm-eba-bnd.xml`, contains the resource references to authentication alias binding for a bundle. If an OSGi application contains an OSGi blueprint binding configuration file when it is deployed as an asset, the binding configuration file provides the default authentication alias values that are used when binding the resource references.

Parent topic: [Creating OSGi projects](#)

Creating blueprint binding XML files

The blueprint binding XML file, `ibm-eba-bnd.xml`, contains the resource references to authentication alias binding for a bundle. If an OSGi application contains an OSGi blueprint binding configuration file when it is deployed as an asset, the binding configuration file provides the default authentication alias values that are used when binding the resource references.

Procedure

1. Open the blueprint XML file for which you want to create the binding XML file.
2. Click **Open WebSphere Blueprint Bindings Descriptor** to create an OSGi blueprint configuration file that contains the resource references to authentication alias binding for a bundle. The Create New Blueprint Binding File dialog box opens.
3. Click **OK** to create a new binding file. The Blueprint Binding XML editor opens.
4. Click **Add** to add the component assembly and configuration information to your blueprint configuration file:

- Resource Reference

- The `resource-ref` element specifies the logical names of application server resources.

- Authentication Alias

- The authentication alias is used for securing the resource reference.

- Interface

- Identifies the interface for the resource reference. The interface can be any of the following types:
 - Default messaging JMS queues destinations
 - Default messaging JMS topic destinations
 - Data source
 - Generic JMS connection factory
 - Mail session
 - J2C connection factory
 - JMS queue connection factory for the JMS provider of WebSphere® MQ
 - JMS queue destination for WebSphere MQ
 - JMS topic connection factory for WebSphere MQ
 - JMS topic destination for WebSphere MQ
 - Unified JMS connection factory for WebSphere MQ
 - URL configuration

- ID

- Identifies the resource reference.

Parent topic: [Creating blueprint XML files](#)

Creating fragment projects

Fragments extend bundles with resources, classes, and permitted headers, which you can use to customize your bundles.

Procedure

1. Click **File > New > Other > OSGi > OSGi Fragment Project** and then click **Next**. The New OSGi Fragment Project wizard opens.
2. In the **Project name** field, enter the name of your fragment project.
3. Select a **Target runtime** from the drop-down list. Set the target run time to define an installed runtime environment. Run times are used at build time to compile projects.
4. Select a **Fragment host** from the drop-down list. The fragment host links the fragment to its potential bundle hosts.
Click **Finish**.

Results

Your OSGi fragment project is created and your fragment manifest, `META-INF/MANIFEST.MF`, is added to your project. For more information about the fragment manifest file, see [OSGi fragment manifest files](#).

For more information about OSGi fragments, see the following subtopics:

- [OSGi fragments](#)

An OSGi fragment is a Java™ archive file with specific manifest headers that enable it to attach to a specified host bundle or specified host bundles to function. Fragments are treated as part of the host bundles. Relevant definitions of the fragment are merged with the host bundles definitions before the host is resolved, provided the information does not conflict. Fragment dependencies are resolved if possible. If the fragment dependencies cannot be resolved, the fragment does not attach to the host bundle. A fragment cannot have its own class loader or bundle activator. It cannot override the information present in the host bundles. Fragments extend bundles with resources, classes, and permitted headers, which enable you to customize your bundles.

- [Editing fragment manifest files](#)

Parent topic: [Creating OSGi projects](#)

Editing fragment manifest files

Procedure

1. In Enterprise Explorer, expand your fragment project.
2. Double-click **Manifest:<name>**, where *<name>* is the name of your fragment, to open your fragment manifest in the editor.

What to do next

For more information about the OSGi fragment manifest file, see [OSGi fragment manifest files](#).

Parent topic: [Creating fragment projects](#)

Creating composite bundle projects

Composite bundles group bundles into aggregates to ensure consistent behavior. A composite bundle contains bundles or references to bundles outside of the workspace or target platform. A composite bundle ensures consistent behavior from a set of shared bundles at a specific version.

About this task

Important: Applicable edition: Full profile

Procedure

1. Click **File > New > Other > OSGi > OSGi Composite Bundle Project** and then click **Next**. The New OSGi Composite Bundle Project wizard opens.
2. In the **Project name** field, enter the name of your bundle project.
3. Select a **Target runtime** from the drop-down list. Set the target run time to define an installed runtime environment. Run times are used at build time to compile projects. Click **Next**.
4. In the OSGi Bundles Selection Page, select the bundles that you want to include in your composite bundle and then click **Finish**.

Results

Your OSGi composite bundle project is created and your composite bundle manifest, `META-INF/COMPOSITEBUNDLE.MF`, is added to your project. For more information about the composite bundle manifest file, refer to [OSGi composite bundle manifest file](#).

For more information about OSGi composite bundles, see the following subtopics:

- [OSGi composite bundles](#)

Composite bundles group bundles into aggregates to ensure consistent behavior. A composite bundle contains bundles or references to bundles outside of the workspace or target platform. A composite bundle ensures consistent behavior from a set of shared bundles at a specific version.

- [Editing composite bundle manifest files](#)

- [Adding bundles to composite bundles](#)

The bundles that are contained in or referenced by a composite bundle are defined with exact versions.

Parent topic: [Creating OSGi projects](#)

Editing composite bundle manifest files

Procedure

1. In Enterprise Explorer, expand your composite bundle project.
2. Double-click **Manifest:<name>**, where *<name>* is the name of your composite bundle, to open your composite bundle manifest in the editor.

What to do next

For more information about the OSGi composite bundle manifest file, see [OSGi composite bundle manifest file](#).

Parent topic: [Creating composite bundle projects](#)

Adding bundles to composite bundles

The bundles that are contained in or referenced by a composite bundle are defined with exact versions.

Procedure

1. Open your composite bundle manifest file in the editor.
2. On the Overview tab, in the Contained Bundles section, click **Add**. The Select Bundles dialog opens.
3. Select the bundles that you want to add to the composite bundle. To select multiple bundles, hold Ctrl and click the bundles. Click **OK**.
4. Save the manifest file.

What to do next

For more information about the OSGi composite bundle manifest file, see [OSGi composite bundle manifest file](#).

Parent topic: [Creating composite bundle projects](#)

Creating OSGi application projects

An OSGi application project groups a set of bundles to provide a coherent business logic. The application can consist of different bundle types such as web enabled bundles and persistence (JPA) enabled bundles.

Before you begin

[Create a bundle project](#). **Important:** Composite bundles that are contained by an OSGi application are supported in WebSphere® Application Server V8.0 and higher.

About this task

The bundles that are contained by an OSGi application offer services that are isolated by the application. The services are not visible outside the application unless they are configured for export:

- The services can be exported from the application to another application.
- The services can be exported from the application for publication as a web service.
- The application can include web enabled bundles for processing HTTP workloads.

The services are not able to consume services outside of the application unless the services are configured to import the services from outside the application. The imported services can be proxies to other services or can be proxies to remote services.

Procedure

1. Click **File > New > Other > OSGi > OSGi Application Project** and then click **Next**. The New OSGi Application Bundle Project wizard opens.
2. In the **Project name** field, enter the name of your application project. Click **Next**.
3. Select a Target runtime from the drop down list. Set the target runtime to define an installed runtime environment. Runtimes are used at build time to compile projects. Click **Next**.
4. On the OSGi Application page of the wizard, select the bundles that you want grouped by the application project. You can also create additional bundles by clicking **New Bundle**. Click **Finish**.

Results

Your OSGi application project is created in your workspace. Your application manifest file, `META-INF/APPLICATION.MF`, contains metadata that enables the OSGi Framework to process the modular aspects of the bundles. For more information about the application manifest file, see [Application manifest files](#).

What to do next

Now that you have created an application project, you can export the project as an EBA file, publish the application to your server, and add or remove bundles from your OSGi application.

For more information about creating OSGi application projects, see the following subtopics:

- [Editing application manifest files](#)
- [Viewing deployment manifest files](#)

- **Adding bundles to OSGi application projects**
- **Adding composite bundles to OSGi application projects**
- **Adding bundle fragments to OSGi application projects**

Parent topic:Creating OSGi projects

Editing application manifest files

About this task

The OSGi application manifest file contains metadata that enables the OSGi Framework to process the modular aspects of the bundle.

Procedure

1. In Enterprise Explorer, expand your application project.
2. Double-click **Manifest:<name>**, where *<name>* is the name of your application, to open your application manifest in the editor.

What to do next

For more information about the OSGi application manifest file, see the following subtopic:

- [Application manifest files](#)

The application manifest file contains metadata that enables the OSGi Framework to process the modular aspects of the bundles.

Parent topic: [Creating OSGi application projects](#)

Application manifest files

The application manifest file contains metadata that enables the OSGi Framework to process the modular aspects of the bundles.

The following code is an example of the contents of an application manifest file:

```
Manifest-Version: 1.0
Application-Name: OSGi Blog Application
Application-SymbolicName: com.ibm.ws.eba.example.blog.app
Application-Version: 1.0.0
Application-Content: com.ibm.ws.eba.example.blog.api;version=1.0.0,
com.ibm.ws.eba.example.blog.persistence;version=1.0.0,
com.ibm.ws.eba.example.blog.web;version=1.0.0,
com.ibm.ws.eba.example.blog;version=1.0.0,
com.ibm.json.java;version="1.0.0"
Application-ManifestVersion: 1.0
Application-ImportService: com.ibm.ws.eba.counter.Greet
Application-ExportService: com.ibm.ws.eba.example.blog.api.BloggingService
```

An application manifest can contain the following headers:

- Application-SymbolicName

- The unique symbolic name of the OSGi application, using similar package notation to Java™.

- Application-Version

- The version of the application, using OSGi syntax for a bundle version.

- Application-Name

- The name of the application.

- Application-ImportService

- Declares the external dependencies of the bundle that are used by the OSGi Framework for bundle resolution.

Specific versions or version ranges for each service can be declared.

A set of filters for external services that the application consumes. The application manifest must contain the classes that services require. If this header is not specified, none of the required services are imported.

Specify the required services in a comma-separated list with the service interface name first followed by attributes or directives. `<service identifier>;<directives>;<attributes>`

For example: `test.it;filter="some_filter"`

The Application-ImportService header has the following attribute:

- filter

- An OSGi service filter.

- Application-ExportService

- Declares the services that are visible outside the bundle. Any service not declared here has visibility only within the bundle.

A set of filters for external services that the application produces. If this header is not specified, none of the required services are exported.

Specify the exported services in a comma-separated list with the service interface name first followed by attributes or directives. `<service identifier>;<directives>;<attributes>`

For example: `test.it;filter="some_filter"`

The Application-ExportService header has the following attribute:

- filter

- An OSGi service filter.

- Use-Bundle

- A shared bundle that provides at least one package to an application bundle.
A list of bundles or composite bundles to use to satisfy the package dependencies of bundles in the Application-Content list. Each bundle or composite bundle in the Use-Bundle list must provide at least one package to at least one bundle in the Application-Content list. These bundles are provisioned into the shared bundle space at run time. Often, you do not require a Use-Bundle header, but there are some situations where it is useful. You can use it to restrict the level at which sharing is possible. For example, you can ensure that an application uses the same bundle for package imports that it was tested with. Alternatively, you can ensure that two applications use the same bundle for package imports. By setting the restriction at application level, the bundle can remain flexible.

- Application-WebModules

- List of non-OSGi dynamic web projects that are included in the application.
This header is not part of the OSGi standard.

- Application-Content

- A list of composite bundles, bundle fragments, and bundles with the acceptable range of OSGi version specifications that are included in the application. **Tip:** When including a bundle fragment in the Application-Content list, ensure that you include the host bundle for the fragment.

The format is a comma-separated list of module declarations, where each module declaration uses the following format: `<module identifier>;<directives>;<attributes>`

Typically, the module identifier is the symbolic name of a bundle. To reference a resource that is not a bundle, the module identifier is the path relative to the OSGi application root.

The Application-Content header has the following attribute:

- version

- The version of the module is specified using OSGi syntax for a version range. Specify the minimum version of the application followed by the maximum version to which the application can be upgraded. For example, "[1.0.0,2.0.0)" means version 1.0.0 and all later versions up to, but excluding, version 2.0.0.

The Application-Content header defines the important applications that compose the business services, but it does not define the full list of bundles in the application. If a bundle that is listed in the content uses a package that is not included in the application, a dependency analysis is performed and any missing bundles are included. These bundles cannot provide services external to the application and cannot have security applied to them. Bundles that are included using this mechanism are shared.

Parent topic: [Editing application manifest files](#)

Viewing deployment manifest files

Before you begin

1. [Create an OSGi application project.](#)
2. [Add the OSGi application to a server instance.](#)
3. Start the server.
4. [Import the deployment manifest file.](#)

About this task

The deployment manifest, `META-INF/DEPLOYMENT.MF`, specifies all the bundles that make up the application, including bundles that are required following dependency analysis. The deployment manifest specifies the actual version of each bundle that is used in the application. It is created automatically when an EBA asset is installed and it ensures that each time an application server starts, the bundles that make up the application are the same.

Note: You cannot edit a deployment manifest file directly. The deployment manifest file is automatically generated when you install your OSGi application on the server.

Procedure

1. In Enterprise Explorer, expand your application project.
2. Expand META-INF and then double-click **DEPLOYMENT.MF** to open your deployment manifest in the editor.

Results

For more information about the OSGi deployment manifest file, see the following subtopic:

- [Deployment manifest files](#)

The deployment manifest, `META-INF/DEPLOYMENT.MF`, specifies all the bundles that make up the application, including bundles that are required following dependency analysis. The deployment manifest specifies the actual version of each bundle that is used in the application. It is created automatically when an EBA asset is installed and it ensures that each time an application server starts, the bundles that make up the application are the same.

Parent topic: [Creating OSGi application projects](#)

Deployment manifest files

The deployment manifest, `META-INF/DEPLOYMENT.MF`, specifies all the bundles that make up the application, including bundles that are required following dependency analysis. The deployment manifest specifies the actual version of each bundle that is used in the application. It is created automatically when an EBA asset is installed and it ensures that each time an application server starts, the bundles that make up the application are the same.

After an application is installed, the version of a bundle can be updated by configuring the EBA asset.

The following code is an example of the contents of a deployment manifest file:

```
Manifest-Version: 1.0
Deployed-Content: bundle1;deployed-version=1.0.0.qualifier
Application-SymbolicName: bundle.app
Application-Version: 1.0.0.qualifier
Import-Package: javax.servlet.jsp;version="2.0.0", javax.persistence;version="0.0.0", javax.servlet.http;version="2.5.0", javax.servlet;version="2.5.0", javax.servlet.jsp.el;version="2.0.0", javax.servlet.jsp.tagext;version="2.0.0", javax.el;version="2.0.0"
```

A deployment manifest contains the following headers:

- Manifest-Version

- A version number for the manifest format.

- Application-SymbolicName

- The unique symbolic name of the application, which uses similar package notation to Java™. This symbolic name matches the `Application-SymbolicName` value in the application manifest.

- Application-Version

- The version of the application, which uses OSGi syntax for a bundle version. This version matches the `Application-Version` value in the application manifest.

- Deployed-Content

- A comma-separated list of the symbolic names of the bundles and the exact versions to be used. The list includes all the bundles that are listed in the `Application-Content` header in the application manifest, and bundles that are imported by dependency analysis. Non-OSGi module types are included using the symbolic name of the converted bundle.

The `Deployed-Content` header has the following directive:

- deployed-version

- The exact version of the bundle, which is specified by using OSGi syntax for a version.

- Deployed-Use-Bundle

- A list of bundles or composite bundles that satisfy the package dependencies of bundles in the `Deployed-Content` list. Each element in the `Deployed-Use-Content` list must provide at least one package to at least one bundle in the `Deployed-Content` list. The `Deployed-Use-Bundle` list is an exact subset of the `Use-Bundle` list. These bundles are loaded into the shared bundle space at run time. An administrator can update bundles that are mapped into the `Deployed-Use-Bundle` list from the `Use-Bundle` list after application deployment.

- Provision-Bundle

- A list of additional bundles and composite bundles that are required as a result of resolving the OSGi application. Each bundle or composite bundle is loaded into the shared bundle space at run time; however, they might not be required. An administrator cannot update bundles in the `Provision-Bundle` list after application deployment.

- Import-Package

- A list of the packages that the bundles in the `Deployed-Content` list consume from the bundles and composite bundles in the `Deployed-Use-Bundle` and `Provision-Bundle` lists. For packages that are consumed from the

Deployed-Use-Bundle list, the package import has `!bundle-symbolic-name` and `!bundle-version` attributes.

Parent topic: [Viewing deployment manifest files](#)

Adding bundles to OSGi application projects

About this task

Bundles can be added to an application as use bundles or application bundles.

- [Adding bundles to an application as a use bundle](#)
- [Adding bundles to an application as an application bundle](#)

Parent topic: [Creating OSGi application projects](#)

Adding bundles to an application as a use bundle

About this task

Bundles are added to the Use-Bundle list to satisfy the package dependencies of bundles in the Application-Content list. Each bundle or composite bundle in the Use-Bundle list must provide at least one package to at least one bundle in the Application-Content list. These bundles are provisioned into the shared bundle space at run time. Provisioning is the process of getting bundles from the repositories.

Learn more about use bundles: Shared bundles are not application-specific. A single instance of a package from a shared bundle can be used by many applications. Shared bundles cannot import packages or services from application bundles. Shared bundles in an application must be provided by reference rather than contained directly in an application. A use bundle is a shared bundle that provides at least one package to an application bundle. Use bundles are referenced in the application manifest in the Use-Bundle header.

Procedure

1. In Enterprise Explorer, expand your OSGi application project and then expand **META-INF**.
2. Double-click **APPLICATION.MF** to open it in the editor.
3. In the Shared Bundle Affinity section, click **Add** and then select the bundle from the list. Click **OK**.

Adding bundles to an application as an application bundle

About this task

Add bundles to the Application-Content list to include the bundles in the application.

Learn more about application bundles: Application bundles are bundles that you create specifically for your application. They are instance-specific and are not shared with other applications. They are referenced in the application manifest in the Application-Content header.

Procedure

1. In Enterprise Explorer, expand your OSGi application project and then expand **META-INF**.
2. Double-click **APPLICATION.MF** to open it in the editor.
3. In the Contained Bundles section, click **Add** and then select the bundle from the list. Click **OK**.

Adding composite bundles to OSGi application projects

About this task

Composite bundles can be added to an application as use bundles or application bundles.

- [Adding composite bundles to an application as a use bundle](#)
- [Adding composite bundles to an application as an application bundle](#)

Parent topic: [Creating OSGi application projects](#)

Adding composite bundles to an application as a use bundle

About this task

Composite bundles are added to the Use-Bundle list to satisfy the package dependencies of bundles in the Application-Content list. Each bundle or composite bundle in the Use-Bundle list must provide at least one package to at least one bundle in the Application-Content list. These bundles are provisioned into the shared bundle space at run time.

Provisioning is the process of getting bundles from the repositories.

Learn more about use bundles: Shared bundles are not application-specific. A single instance of a package from a shared bundle can be used by many applications. Shared bundles cannot import packages or services from application bundles. Shared bundles in an application must be provided by reference rather than contained directly in an application. A use bundle is a shared bundle that provides at least one package to an application bundle. Use bundles are referenced in the application manifest in the Use-Bundle header.

Procedure

1. In Enterprise Explorer, expand your OSGi application project and then expand **META-INF**.
2. Double-click **APPLICATION.MF** to open it in the editor.
3. In the Shared Bundle Affinity section, click **Add** and then select the composite bundle from the list. Click **OK**.

Adding composite bundles to an application as an application bundle

About this task

Add composite bundles to the Application-Content list to include the composite bundles in the application.

Learn more about application bundles: Application bundles are bundles that you create specifically for your application. They are instance-specific and are not shared with other applications. They are referenced in the application manifest in the Application-Content header.

Procedure

1. In Enterprise Explorer, expand your OSGi application project and then expand **META-INF**.
2. Double-click **APPLICATION.MF** to open it in the editor.
3. In the Contained Bundles section, click **Add** and then select the composite bundle from the list. Click **OK**.

Adding bundle fragments to OSGi application projects

Procedure

1. In Enterprise Explorer, expand your OSGi application project and then expand **META-INF**.
2. Double-click **APPLICATION.MF** to open it in the editor.
3. In the Contained Bundles section, click **Add** and then select the fragment from the list. Click **OK**.

Parent topic: [Creating OSGi application projects](#)

Importing and configuring OSGi applications

Learn how to import EBA files as OSGi applications, bundle JAR files as OSGi bundles, fragment JAR files as fragments, and CBA files as OSGi composite bundles. Learn also how to import deployment manifest files.

- **Importing OSGi application projects**

You can import an EBA file as an OSGi application project. An enterprise bundle archive (EBA) defines a set of OSGi bundles that are deployed as a single OSGi application, and that are isolated from other OSGi applications.

- **Importing OSGi bundle projects**

You can import a bundle JAR file as an OSGi bundle project. An OSGi bundle is a Java™ archive file that contains Java code, resources, and a manifest that describes the bundle and its dependencies.

- **Importing OSGi composite bundle projects**

You can import a CBA archive file as an OSGi composite bundle project.

- **Importing OSGi fragment projects**

You can import a fragment JAR file as an OSGi fragment project.

- **Importing deployment manifest files**

When an application is deployed to the server, a deployment manifest file is generated. It contains the exact set of bundles, at the specific versions, that are deployed as part of that application. If you want to run the same application on another server, or view the contents of the deployment manifest file, you can import the deployment manifest into your workspace.

Parent topic: [Developing OSGi applications](#)

Importing OSGi application projects

You can import an EBA file as an OSGi application project. An enterprise bundle archive (EBA) defines a set of OSGi bundles that are deployed as a single OSGi application, and that are isolated from other OSGi applications.

Procedure

1. Click **File > Import > OSGi > OSGi Application EBA**. Click **Next**.
2. Click **Browse** to select the EBA file.
3. Select the **Target runtime**. **Note:** If you select a target run time that differs from your workspace target platform, your projects might import with errors.
4. Select the bundles that you want to import from the **Bundles to include** list.
5. Follow the wizard prompts.

Results

The EBA file is imported into your workspace as an application project and the artifacts inside the EBA file are imported as bundle, composite bundle, or fragment projects, depending on the type of artifact.

Parent topic: [Importing and configuring OSGi applications](#)

Importing OSGi bundle projects

You can import a bundle JAR file as an OSGi bundle project. An OSGi bundle is a Java™ archive file that contains Java code, resources, and a manifest that describes the bundle and its dependencies.

Procedure

1. Click **File > Import > OSGi > OSGi Bundle or Fragment**. Click **Next**.
2. Click **Browse** to select the Bundle JAR file.
3. Select the **Target runtime**. **Note:** If you select a target run time that differs from your workspace target platform, your projects might import with errors.
4. Follow the wizard prompts.

Results

The bundle JAR file is imported into your workspace as a bundle project.

Parent topic: [Importing and configuring OSGi applications](#)

Importing OSGi composite bundle projects

You can import a CBA archive file as an OSGi composite bundle project.

About this task

Important: Applicable edition: Full profile

Procedure

1. Click **File > Import > OSGi > OSGi Composite Bundle**. Click **Next**.
2. Click **Browse** to select the CBA file.
3. Select the **Target runtime**. **Note:** If you select a target run time that differs from your workspace target platform, your projects might import with errors.
4. In the Bundles to include list, select the bundles and fragments that you want to import into your workspace.
5. Follow the wizard prompts.

Results

The composite bundle project and selected bundle projects are imported into your workspace.

Parent topic: [Importing and configuring OSGi applications](#)

Importing OSGi fragment projects

You can import a fragment JAR file as an OSGi fragment project.

Procedure

1. Click **File > Import > OSGi > OSGi Bundle or Fragment**. Click **Next**.
2. Click **Browse** to select the fragment JAR file.
3. Select the **Target runtime**. **Note:** If you select a target run time that differs from your workspace target platform, your projects might import with errors.
4. Follow the wizard prompts.

Results

The fragment JAR file is imported into your workspace as a fragment project.

Parent topic: [Importing and configuring OSGi applications](#)

Importing deployment manifest files

When an application is deployed to the server, a deployment manifest file is generated. It contains the exact set of bundles, at the specific versions, that are deployed as part of that application. If you want to run the same application on another server, or view the contents of the deployment manifest file, you can import the deployment manifest into your workspace.

Before you begin

1. [Create an OSGi application project.](#)
2. [Add the OSGi application to a server instance.](#)
3. Start the server

Procedure

1. In the Servers view, expand your server instance and right-click your OSGi application and select **View Deployment Manifest**. The Deployment manifest file opens in the editor.
2. Click **Import**. Your deployment manifest file is imported into the META-INF folder of your application.

Parent topic: [Importing and configuring OSGi applications](#)

Exporting OSGi applications

Learn how to export OSGi applications, OSGi bundles, OSGi fragments, and OSGi composite bundles. Learn also how to export deployment manifest files.

- **Exporting OSGi application projects**

- **Exporting OSGi bundle projects**

You can export your OSGi bundle project as a bundle JAR file. An OSGi bundle is a Java™ archive file that contains Java code, resources, and a manifest that describes the bundle and its dependencies.

- **Exporting OSGi composite bundle projects**

You can export your OSGi composite bundle project as a CBA archive file.

- **Exporting OSGi fragment projects**

You can export your OSGi fragment project as a JAR file.

- **Exporting deployment manifest files**

When an application is deployed to the server, a deployment manifest file is generated. It contains the exact set of bundles, at the specific versions, that are deployed as part of that application. If you want to run the same application on another server, you can export the deployment manifest as part of the OSGi application EBA and import it to the new server to ensure that the same set of bundles with the same versions are deployed.

Parent topic: [Developing OSGi applications](#)

Exporting OSGi application projects

About this task

You can export your OSGi application project as an EBA file. An enterprise bundle archive (EBA) defines a set of OSGi bundles, composite bundles, and fragments that are deployed as a single OSGi application, and that are isolated from other OSGi applications.

Procedure

1. In Enterprise Explorer, right-click your OSGi application project and then select **Export > OSGi Application EBA**. The Export wizard opens.
2. In the **To EBA file** field, specify the location and name of the EBA file to which you want to export the application project EBA file.
3. In the **Bundles to include** section, select the bundles that you want to export. Note: By default the highest level of a bundle is displayed. If you want to include a different version of a bundle, click the bundle then select the version button. The version selection dialog presents the bundle versions that are available.
4. Click **Finish**. Your project is exported as an EBA file to the location you specified.

Parent topic: [Exporting OSGi applications](#)

Exporting OSGi bundle projects

You can export your OSGi bundle project as a bundle JAR file. An OSGi bundle is a Java™ archive file that contains Java code, resources, and a manifest that describes the bundle and its dependencies.

Before you begin

[Create an OSGi bundle project.](#)

Procedure

1. In Enterprise Explorer, click your OSGi bundle project and then click **File > Export > OSGi > OSGi Bundle or Fragment** and then click **Next**. The Export wizard opens.
2. In the **To JAR file** field, specify the location to which you want to export the bundle project JAR file. Click **Finish**.

Parent topic: [Exporting OSGi applications](#)

Exporting OSGi composite bundle projects

You can export your OSGi composite bundle project as a CBA archive file.

Before you begin

Important: Applicable edition: Full profile

[Create an OSGi composite bundle project.](#)

Procedure

1. In Enterprise Explorer, click your OSGi composite bundle project and then click **File > Export > OSGi > OSGi Composite Bundle** and then click **Next**. The Export wizard opens.
2. In the **To CBA file** field, specify the location to which you want to export the composite bundle archive file.
3. In the **Bundles to include** list, select the bundles and any dependent bundles and fragments that you want to include in the CBA file. Note: By default the highest level of a bundle is displayed. If you want to include a different version of a bundle, click the bundle then select the version button. The version selection dialog presents the bundle versions that are available.
4. Click **Finish**.

Parent topic: [Exporting OSGi applications](#)

Exporting OSGi fragment projects

You can export your OSGi fragment project as a JAR file.

Before you begin

[Create an OSGi fragment project.](#)

Procedure

1. In Enterprise Explorer, click your OSGi composite bundle project and then click **File > Export > OSGi > OSGi Bundle or Fragment** and then click **Next**. The Export wizard opens.
2. In the **To JAR file** field, specify the location to which you want to export the fragment project JAR file. Click **Finish**.

Parent topic: [Exporting OSGi applications](#)

Exporting deployment manifest files

When an application is deployed to the server, a deployment manifest file is generated. It contains the exact set of bundles, at the specific versions, that are deployed as part of that application. If you want to run the same application on another server, you can export the deployment manifest as part of the OSGi application EBA and import it to the new server to ensure that the same set of bundles with the same versions are deployed.

Procedure

1. In Enterprise Explorer, right-click your OSGi application project and then select **Export > OSGi Application (EBA)**. The Export wizard opens.
2. In the **To EBA file** field, specify the location and name of the EBA file to which you want to export the application project EBA file. Click **Finish**. The deployment manifest is exported as part of the EBA file.

Parent topic: [Exporting OSGi applications](#)

Converting existing projects to OSGi projects

About this task

You can convert your Java™ EE modules, Java projects, and PDE plug-ins to OSGi projects to take advantage of the many [benefits of the OSGi framework](#).

Procedure

1. Launch the conversion dialog. In Enterprise Explorer, right-click your PDE, Java EE module, or EJB project and select **Configure > Convert to OSGi Bundle Project**. This conversion assumes that the project name and location are the same. **Note:** If you are converting a PDE fragment project or a Java EE fragment project to an OSGi fragment project, right-click the fragment and select **Configure > Convert to OSGi Bundle Fragment Project**.
2. Review the information in the conversion dialog.

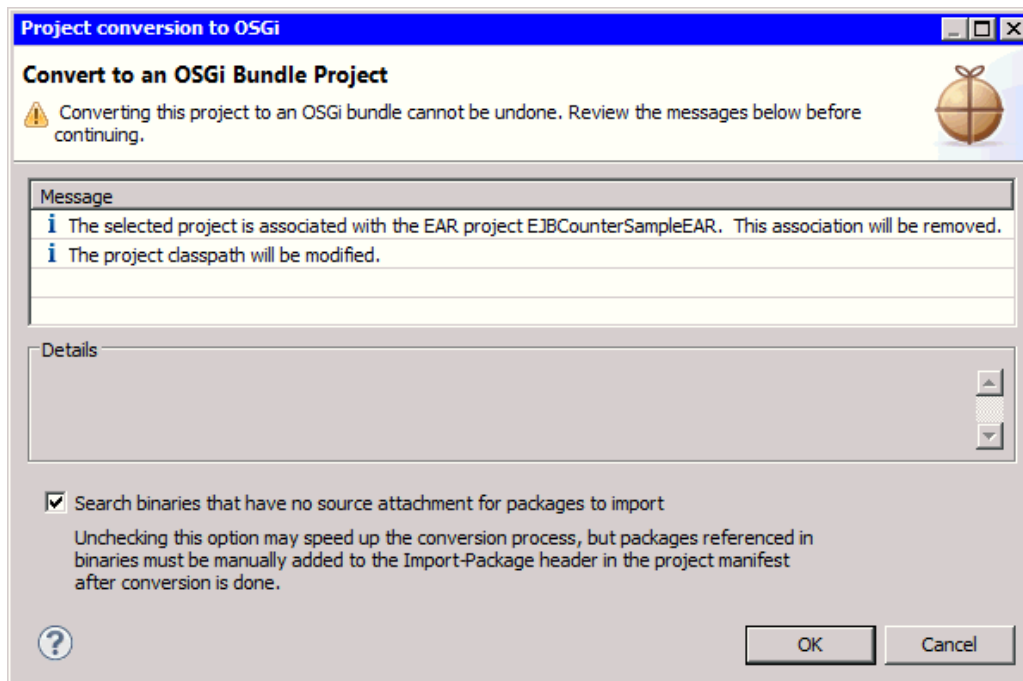
- Messages and Details

- The conversion dialog displays messages about the projects you are converting. If you select a message, if there is additional information it is displayed in the **Details** section.

- Search binaries that have no source attachment for packages to import

- When you convert a project with Java source files, packages that are not satisfied within the project are automatically added to the new bundle manifest file. However, if the source is not available, the option **Search binaries that have no source attachment for packages to import** facilitates the conversion. Note that when you search binary files, there are some instances when packages are not found. For example, the package information for objects used only within method blocks is not found. If necessary, in these cases you can add the import information manually to the manifest. **Search binaries that have no source attachment for packages to import** is selected by default.

The following image shows the conversion dialog with example messages for an EJB conversion.



3. Click **OK** to convert the project.

Results

When your Java EE, EJB, or PDE project is converted to an OSGi bundle, OSGi specific natures, builders, class path entries, and project resources are added to the OSGi application. PDE-specific natures, builders, class path entries, and project resources are removed from the newly converted OSGi application. The project settings are updated to include default preferences for your OSGi application.

The bundle manifest file, MANIFEST.MF, is created or updated to include all of the required bundle properties. An OSGi bundle manifest describes the bundle and bundle dependencies. For more information about the bundle manifest file, see [OSGi bundle manifest file](#). For information on the fragment manifest file, refer to [OSGi fragment manifest files](#).

What to do next

After you convert your PDE or Java EE project into an OSGi bundle project, you can create your business logic, export the OSGi bundle as a JAR file, add the bundle to an OSGi application, or add the bundle to a composite bundle.

Parent topic: [Developing OSGi applications](#)

Adding OSGi support to Maven projects

You can add OSGi support to Maven projects by converting the project to an OSGi bundle project.

Procedure

1. Right-click the Maven project and select **Configure > Convert to OSGi Bundle Project**.
2. A message displays stating that the `pom.xml` file will be modified. Click **OK**.

Results

OSGi support is added to the project. The following files are created or modified:

- A `manifest.mf` file is generated and contains configuration and dependency information from the `pom.xml`. This `manifest.mf` file is updated automatically as you work with your project. Do not edit the `manifest.mf` file.
- The Project Object Model (POM) file, `pom.xml`, is modified.

Parent topic: [Developing OSGi applications](#)

Adding Maven support to OSGi applications

You can add Maven support to OSGi applications by converting the project to a Maven project. The converted project remains an OSGi application but configurations for the bundle, such as its ID, version, and its dependencies are defined in a `pom.xml` file.

Procedure

1. Right-click the OSGi application and select **Configure > Convert to Maven Project**.
2. Complete the Create New POM dialog box. Enter a **Group Id**, **Artifact Id**, and **Version** or accept the defaults. Optionally, enter a **Name** and **Description**. For **Packaging**, accept the default value.
3. Click **Finish**.

Results

The following changes are made to the project:

- A Project Object Model (POM) file, `pom.xml`, is generated for your project and contains all configuration, including the dependencies, from the existing `manifest.mf` file.
- The existing `manifest.mf` file is updated to indicate that the file is generated by the Maven project. Do not edit the generated `manifest.mf` file.

New dependencies that you add to your project are specified in the `pom.xml`. Changes in the `pom.xml` file are automatically generated in the `manifest.mf` file.

What to do next

If your project uses WebSphere® Application Server APIs, you must manually add them as a dependency in your POM file:

1. Install the server APIs in your Maven repository. For instructions, see [Installing the server APIs into the Maven repository](#).
. **Tip:** It is only necessary to install the API to the local repository one time.
2. Add a dependency to the APIs. If you added the API to your Maven repository, right-click the project and select **Maven > Add Dependency**. Select the module; the **Group Id** and **Artifact Id** fields are updated. Set **Scope** to `provided`. Click **OK**.

For details of how to export packages from a JAR file in a Maven bundle, see the following subtopic:

- [Exporting packages from a JAR file in a Maven bundle](#)

If you want to export packages from a JAR file in a Maven bundle, and the Maven project was converted from an OSGi bundle, you must modify the `pom.xml` file.

Parent topic: [Developing OSGi applications](#)

Exporting packages from a JAR file in a Maven bundle

If you want to export packages from a JAR file in a Maven bundle, and the Maven project was converted from an OSGi bundle, you must modify the `pom.xml` file.

Procedure

To export packages from a JAR file in a Maven bundle:

1. In Enterprise Explorer, expand your bundle project.
2. Double-click the `pom.xml` file.
3. Click the **pom.xml** tab.
4. Locate the instructions section (between the `<instructions>` and `</instructions>` tags).
5. Add a classpath element (`<classpath></classpath>`).
6. Between the classpath tags, add the relative of the JAR file. If there are multiple JAR files, list the paths to each JAR file separated by a comma.
7. After the classpath tags, add an Export-Package element (`<Export-Package></Export-Package>`).
8. Between the Export-Package tags, list the JAR file that you want to export. **Tip:** You can use the character `*` as a wildcard.
9. Click **File > Save**.

Parent topic: [Adding Maven support to OSGi applications](#)

Accessing data using Java Persistence API (JPA)

Java™ Persistence API (JPA) is a specification for the persistence of Java objects to relational databases. JPA helps you to manage relational data in your applications

About this task

The following subtopics describe the ways in which you can access data by using JPA in OSGi bundles:

- [Accessing data by using JPA inside of the bundle](#)
- [Accessing data using JPA in a different bundle](#)

Parent topic: [Developing OSGi applications](#)

Accessing data by using JPA inside of the bundle

About this task

Accessing data by using JPA inside of the bundle assumes that you are working in a bundle that is treated as a Java™ EE module on the server. For example, an OSGi web application bundle that has a JPA facet enabled and contains entities and the logic to access, consume, and display data by using JPA. Another example is a web project that contains JPA entities and the logic to consume data by using JPA that is published in an OSGi application rather than as part of an EAR deployment.

You need to configure your OSGi bundle and add the data sources to WebSphere® Application Server before you can access the JPA persistence units.

Tip: Ensure that your JPA persistence file contains references to a Java Transaction API (JTA) and non-JTA data source. JPA has two transactional patterns for accessing a data source:

- **jta-data-source**

- The Java Transaction API (JTA) resource pattern depends on global transactions. The JTA resource pattern is typically used within the scope of an Enterprise JavaBeans (EJB) session facade. This configuration allows the session bean to control transaction and security contexts while JPA handles the persistence mappings. In this case, the application does not use the EntityTransaction interface but relies on the EntityManager enlisted with the global transaction when it is accessed.

- **non-jta-data-source**

- The non-JTA resource pattern is used to deal with a single resource in the absence of global transactions. The non-JTA resource pattern is typically used within the scope of a web application or an application client. The application controls the transaction with the data source with the EntityTransaction interface.

In `persistence.xml` files for an OSGi application, the `jta-data-source` and `non-jta-data-source` elements access the data sources through a Java Naming and Directory Interface (JNDI) lookup, a JNDI lookup to the service registry, or through Blueprint.

If the JTA and non-JTA data sources are not configured in the `persistence.xml` file, the default JTA and non-JTA data sources configured for the server are used. By default the values are null. Some JPA entity features require a non-JTA data source to be specified. For example, automatic entity identity generation.

Procedure

1. Add non-JTA data sources to `persistence.xml`:

- A. Open `persistence.xml` in the editor.
- B. In the Overview section look at the components list and select your entity to display the details of your entity.
- C. In the Non-JTA **data source** field, type the global JNDI name of a non-JTA data source. For example,

```
osgi:service/javax.sql.DataSource/(osgi.jndi.service.name=jdbc/blogdbnojta).
```

2. Add JTA data sources to `persistence.xml`:

- A. Open `persistence.xml` in the editor.
- B. In the Overview section look at the components list and select your entity to display the details of your entity.
- C. In the JTA **data source** field, type the global JNDI name of the data source. For example,

```
osgi:service/javax.sql.DataSource/(osgi.jndi.service.name=jdbc/blogdb).
```

3. Add a JDBC provider to the WebSphere Application Server administrative console:

- A. Switch to the Servers view.

- B. Right-click your server instance and select **Start**.
 - C. Right-click your server instance and select **Administration > Run Administrative Console** to open the administrative console.
 - D. Click **Resources > JDBC > JDBC providers**.
 - E. Click **New** in the JDBC providers page. The Create a data source wizard opens.
 - F. Follow the instructions in the wizard to create the JDBC provider.
 - G. Save your changes,
4. Add data source definitions to the WebSphere Application Server administrative console:
 - A. In the administrative console, click **Resources > JDBC > Data sources** to open the Data sources page in the console.
 - B. In the Data sources page, click **New** to create a data source definition with a JNDI name set to the JTA connection definition specified in your `persistence.xml`. For example, `jdbc/blogdb`.
 - C. In the Data sources page, click **New** to create another data source definition with a JNDI name set to the non-JTA connection definition specified in your `persistence.xml`. For example, `jdbc/blogdbnojta`.
 - D. When the data source for the non-JTA connection is created, click the definition for the non-JTA connection in the Data sources page of the admin console. The Configuration page opens.
 - E. In the Additional Properties section, click **WebSphere Application Server data source properties**.
 - F. Select **Non-transactional data source**. Within the application server, the use of the `<non-jta-data-source>` element requires a special configuration for a non-transactional data source. Data sources that are configured for the application server do not function as a `<non-jta-data-source>` because all data sources that are configured by the application server are automatically enlisted with the current transactional context. To prevent this automatic enlistment, add a data source custom property `nonTransactionalDataSource=true`.

Results

The `persistence.xml` and WebSphere Application Server are configured to access JPA in an OSGi bundle.

Parent topic: [Accessing data using Java Persistence API \(JPA\)](#)

Accessing data using JPA in a different bundle

About this task

You need to configure your JPA bundle and add the data sources to WebSphere® Application Server before you can access JPA persistence units from another bundle. For example, a web application bundle that consumes JPA entities and displays and manipulates data. **Tip:** Ensure that your JPA persistence file contains references to a Java™ Transaction API (JTA) and non-JTA data source.

JPA has two transactional patterns for accessing a data source:

- jta-data-source

- The Java Transaction API (JTA) resource pattern depends on global transactions. The JTA resource pattern is typically used within the scope of an Enterprise JavaBeans (EJB) session facade. This configuration allows the session bean to control transaction and security contexts while JPA handles the persistence mappings. In this case, the application does not use the EntityTransaction interface but relies on the EntityManager enlisted with the global transaction when it is accessed.

- non-jta-data-source

- The non-JTA resource pattern is used to deal with a single resource in the absence of global transactions. The non-JTA resource pattern is typically used within the scope of a web application or an application client. The application controls the transaction with the data source with the EntityTransaction interface.

In `persistence.xml` files for an OSGi application, the `jta-data-source` and `non-jta-data-source` elements access the data sources through a Java Naming and Directory Interface (JNDI) lookup, a JNDI lookup to the service registry, or through Blueprint.

If the JTA and non-JTA data sources are not configured in the `persistence.xml` file, the default JTA and non-JTA data sources configured for the server are used. By default the values are null. Some JPA entity features require a non-JTA data source to be specified. For example, automatic entity identity generation.

Procedure

1. Ensure that the entity and entity controller packages are added to `manifest.mf` as export packages:
 - A. Double-click **Manifest: <project_name>**, where `<project_name>` is the name of your JPA bundle project. The bundle manifest opens in the editor.
 - B. Switch to the Runtime tab. Verify that the entities and entity controller packages are added as export packages.
2. Add non-JTA data sources to `persistence.xml`:
 - A. Open `persistence.xml` in the editor.
 - B. In the components list, select your entity to display the details of your entity.
 - C. In the Non-JTA data source field, type the global JNDI name of a non-datasource. For example,

```
osgi:service/javax.sql.DataSource/(osgi.jndi.service.name=jdbc/blogdbnojta).
```
3. Modify the `getEntityManager()` method. By modifying the `getEntityManager()` method, you are configuring the JNDI lookup for the `EntityManagerFactory` service since the persistence unit in a JPA bundle is not in a Java EE environment.

You can also configure the JNDI lookup by making the entity manager bean a blueprint managed bean. For more information, see [JPA and OSGi Applications](#).

 - A. Open the entity manager bean in the editor.

B. Locate `getEntityManager()` and modify it as follows:

```
private EntityManager getEntityManager() {
    try {
        emf = (EntityManagerFactory) new
InitialContext().lookup("osgi:service/javax.persistence.EntityManagerFactory/(osgi.unit.name=jpaBundle)");
    } catch (NamingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return emf.createEntityManager();
}
```

Important: In the following line of code, ensure that the persistence unit matches the persistence unit in

```
persistence.xml.emf = (EntityManagerFactory) new
InitialContext().lookup("osgi:service/javax.persistence.EntityManagerFactory/(osgi.unit.name=jpaBundle)");
```

4. Fix generated errors on `InitialContext` and `NamingException`:

A. Switch to the Markers view.

B. For each error, right-click the error and select **QuickFix**. Follow the instructions in the wizard to import the required packages.

5. Add a JDBC provider to the WebSphere Application Server administrative console:

A. Switch to the Servers view.

B. Right-click your server instance and select **Start**.

C. Right-click your server instance and select **Administration > Run Administrative Console** to open the administrative console.

D. Click **Resources > JDBC > JDBC providers**.

E. Click **New** in the JDBC providers page. The Create a data source wizard opens.

F. Follow the instructions in the wizard to create the JDBC provider.

G. Save your changes,

6. Add data source definitions to the WebSphere Application Server admin console:

A. In the administrative console, click **Resources > JDBC > Data sources** to open the data sources page in the console.

B. In the Data sources page, click **New** to create a data source definition with a JNDI name set to the JTA connection definition specified in `persistence.xml`. For example, `jdbc/blogdb`.

C. In the Data sources page, click **New** to create another data source definition with a JNDI name set to the non-JTA connection definition specified in `persistence.xml`. For example, `jdbc/blogdbnojta`.

D. After the data source for the non-JTA connection is created, click the definition for the non-JTA connection in the Data sources page of the admin console. The Configuration page opens.

E. In the Additional Properties section, click **WebSphere Application Server data source properties**.

F. Select **Non-transactional data source**. Within the application server, the use of the `<non-jta-data-source>` element requires a special configuration for a non-transactional data source. Data sources that are configured for the application server do not function as a `<non-jta-data-source>` because all data sources that are configured by the application server are automatically enlisted with the current transactional context. To prevent this automatic enlistment, add an additional data source custom property `nonTransactionalDataSource=true`.

Results

You can now consume JPA entities, display, and manipulate JPA data, where a JPA bundle is accessed from a web application bundle.

Parent topic: [Accessing data using Java Persistence API \(JPA\)](#)

Deploying OSGi applications

Learn how to deploy your OSGi applications, and composite bundles.

About this task

The following subtopics describe how to deploy OSGi applications:

- [Deploying OSGi application projects](#)
- [Deploying OSGi composite bundle projects](#)
- [Removing OSGi business level application from WebSphere Application Server](#)

Application artifacts can sometimes remain in a server instance after a failed publish or removal step. The remaining artifacts can block subsequent attempts to publish an application. Usually, the publishing tool rolls back unsuccessful operations but occasionally you might need to manually remove artifacts.

Parent topic: [Developing OSGi applications](#)

Deploying OSGi application projects

Before you begin

1. [Create an OSGi application.](#)
2. Install WebSphere® Application Server. **Tip:** You can deploy OSGi applications on WebSphere Application Server Version 7, Version 8.0, and Version 8.5.

Learn more about installing WebSphere Application Server Version 7.0:

- A. Open the IBM® Installation Manager.
- B. Click **Install**. The Install Packages page opens.
- C. In the package list, select **IBM WebSphere Application Server Version 7.0 Test Environment**, then click **Next**.
- D. Read the license agreements. Accept the license agreements then click **Next**.
- E. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 7.0.
- F. In the Features list, ensure that you select **OSGi Applications** under **IBM WebSphere Application Server Version 7.0 Feature Pack for OSGi Applications and Java Persistence API 2.0**.

Learn more about installing WebSphere Application Server Version 8.0:

- A. Open the IBM Installation Manager.
- B. Click **Install**. The Install Packages page opens.
- C. In the package list, select **Application Server Version 8.0.0.0**, then click **Next**.
- D. Read the license agreements. Accept the license agreements then click **Next**.
- E. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.0.

Learn more about installing WebSphere Application Server Version 8.5:

- A. Open the IBM Installation Manager.
- B. Click **Install**. The Install Packages page opens.
- C. In the package list, select **Application Server Version 8.5.0.0**, then click **Next**.
- D. Read the license agreements. Accept the license agreements then click **Next**.
- E. Follow the instructions in the Installation Manager to install WebSphere Application Server Version 8.5.

Important: If you use a different test environment, the steps to deploy your application differ from the steps that are provided in this topic.

About this task

To deploy your OSGi application to a server:

1. [Add your OSGi application to a server instance.](#)
2. [Run your application on the server.](#)

Restriction: If you updated the container path of the target definition on the Server Preferences the Target Platform Preference is overridden. For more information, see [Bug 300861](#).

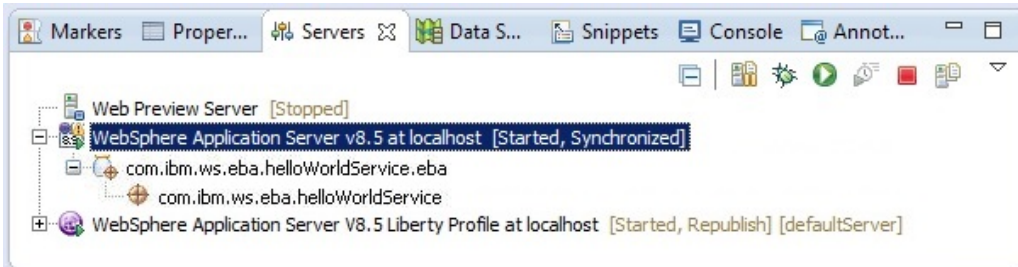
Parent topic: [Deploying OSGi applications](#)

Adding your OSGi application to a server instance

Procedure

1. In the Servers view (**Window > Show View > Servers**), right-click your server instance and select **Add and Remove**.
The Add and Remove dialog opens.
2. In the list of Available resources, select your OSGi application project and then click **Add** to add it to the list of Configured resources.
3. Click **Finish**.

Results



When you add your OSGi application project to the server, the server checks any package dependencies. If your OSGi application has package dependencies outside of your application project, the dependencies are resolved against any configured bundle repositories. The application is started after all bundle dependencies are verified.

Running your application on the server Procedure

In the Servers view (**Window > Show View > Servers**), right-click your server instance and then select **Start**. Your OSGi application is started after the publish completes.

Deploying OSGi composite bundle projects

Before you begin

Important: Applicable edition: Full profile

1. [Create a composite bundle project](#).
2. Install WebSphere® Application Server:
 - A. Open the IBM® Installation Manager.
 - B. Click **Install**. The Install Packages page opens.
 - C. In the package list, select the WebSphere Application Server version that you want to install, then click **Next**.
 - D. Read the license agreements. Accept the license agreements then click **Next**.
 - E. Follow the instructions in the Installation Manager to install the WebSphere Application Server product.

Important: If you use a different test environment, the steps to deploy your application differ from the steps provided in this topic.

About this task

Important: Updating a deployed CBA does not automatically republish the EBAs that reference the CBA.

- If the publishing setting for your server is set to run the resources in your workspace, restart the application on the server in order for the EBA to recognize the changes in the CBA.
- If the publishing setting for your server is set to run the resources on the server, remove the application from the server and then republish the application.

For more information, see [Checking the update status of an OSGi composition unit](#).

To deploy your OSGi composite bundle to a server:

Procedure

1. [Add your composite bundle to a server instance](#).
2. [Run your application on the server](#).

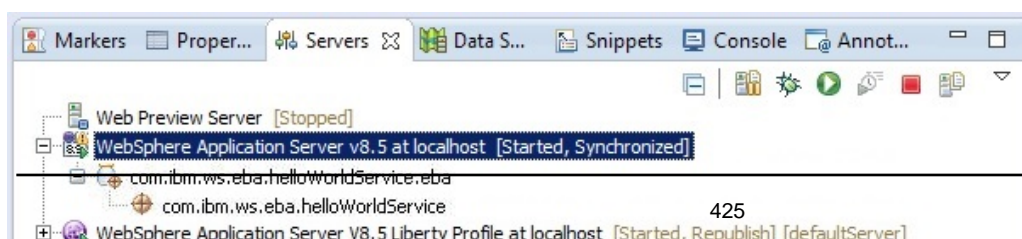
Parent topic: [Deploying OSGi applications](#)

Add your OSGi composite bundle to a server instance

Procedure

1. In the Servers view (**Window > Show View > Servers**), right-click your server instance and select **Add and Remove**. The Add and Remove dialog opens.
2. In the list of Available resources, select your OSGi composite bundle project and then click **Add** to add it to the list of configured resources.
3. Click **Finish**.

Results



When you add your OSGi composite bundle project to the server, the server checks any package dependencies. If your composite bundle has package dependencies outside of your composite bundle project, the dependencies are resolved against any configured bundle repositories. The composite is started after all bundle dependencies are verified.

Run your application on the server Procedure

If the server is not running, go to the Servers view (**Window > Show View > Servers**), right-click your server instance, and then select **Start**. Your OSGi application is started after the publish completes.

Removing OSGi business level application from WebSphere Application Server

Application artifacts can sometimes remain in a server instance after a failed publish or removal step. The remaining artifacts can block subsequent attempts to publish an application. Usually, the publishing tool rolls back unsuccessful operations but occasionally you might need to manually remove artifacts.

Before you begin

Important: Applicable edition: Full profile

Ensure that your server is Started.

Procedure

1. Open the administrative console:
 - A. In the Servers view (**Window > Show View > Servers**), right-click your server instance and select **Administration > Run Administrative Console**. The WebSphere® Application Server Admin Console opens.
2. Remove the composition unit that is associated with the business level application (BLA) from WebSphere Application Server:
 - A. In the task list, expand **Applications > Application Types** and then select **Business-level applications**.
 - B. In the Business-level applications list, click the application that you want to remove. The General Properties page opens.
 - C. In the Deployed assets section, select the assets for the application that you want to remove and then click **Delete**.
Click **OK** to confirm the deletion and then click **Save** to save the change to the master configuration.
3. Remove the BLA from WebSphere Application Server:
 - A. Select the BLA that you want to remove and then click **Delete**. Click **OK** to confirm the deletion and then click **Save** to save the change to the master configuration.
4. Remove the enterprise business asset (EBA) from WebSphere Application Server:
 - A. In the task list, expand **Applications > Application Types** and then select **Assets**.
 - B. Select the EBA that you want to remove and then click **Delete**. Click **OK** to confirm the deletion and then click **Save** to save the change to the master configuration.
5. Remove external bundle repositories:
 - A. In the task list, expand **Environment > OSGi bundle repositories** and then select **External bundle repositories**.
 - B. Select the bundle repository that you want to remove and then click **Delete**. Click **Save** to save the change to the master configuration.
6. Remove the BLA from the server view:
 - A. In the Servers view, right-click the BLA that you want to remove and then click **Remove**.
 - B. Right-click your server instance and click **Restart**.

Parent topic: [Deploying OSGi applications](#)

Using independent Java™ archives (JARs) with OSGi applications

If you want your OSGi application to use packages in JAR files that were developed independently of your application, there are two primary approaches:

- [Including a JAR file in an OSGi application](#)
- [Depending on a JAR file without including it in an OSGi application](#)

In both of these approaches, the JAR file must first be in OSGi bundle format.

There are some advantages and disadvantages to each of these approaches that you should consider. *Table 1.*

	Advantages	Disadvantages
Including the JAR file in an OSGi application	Application is more self-contained Less configuration to use the JAR file If you have access to the source, you might want to modify the packages that you are including. In this case, there is less incentive to keep the archive separate	If there are changes to the archive after the application is deployed, you will need to uninstall and reinstall the whole application on the server
Depending on the JAR file without including it in an OSGi application	Application is more lightweight Keeps libraries and code that is developed independently of your application separate If there are non-breaking changes to the JAR file, you do not need to uninstall and reinstall a deployed OSGi application	Application is not self-contained Configuration is required in the target runtime information of the development workspace Configuration is required on the server

For more information about using independent JAR files with OSGi applications, see the following subtopics:

- [Creating an OSGi bundle from a JAR file](#)
- [Including a JAR file in an OSGi application](#)
- [Depending on a JAR file without including it in an OSGi application](#)

Parent topic: [Developing OSGi applications](#)

Creating an OSGi bundle from a JAR file

About this task

If you have an existing JAR file that you want to use as an OSGi bundle, you can create a bundle from the JAR file. The JAR file can be in your file system or in a project in your workspace. The following two tasks describe the steps for each scenario.

Parent topic: [Using independent Java archives \(JARs\) with OSGi applications](#)

Creating an OSGi bundle from a JAR file in the file system

Procedure

1. Access the Import wizard. Click **File > Import**.
2. Choose **OSGi > Java Archive into an OSGi Bundle**. Click **Next**.
3. Choose the JAR file. In the wizard, for **JAR file**, click **Browse** then browse to the location on your hard disk for the JAR file that you want to import.
4. Choose a bundle or create a new bundle to add the JAR file to. In the wizard, if you have an existing bundle in your workspace that you want to add the JAR file to, select the bundle from the drop-down menu. If you want to create a bundle for the JAR file, click **New Bundle**. If you are creating a new bundle, follow the steps of the wizard. When done, click **Finish** to return to the import wizard. The **JAR file** and **Bundle** fields are now populated.
5. Click **Next** to proceed to the packages screen of the wizard.
6. From the **Packages** list, select the packages in the JAR file that you want to be added to the Export-Package header in the bundle MANIFEST.MF file.
7. Click **Finish**. A bundle is now created based on the JAR file, and export entries for selected packages in the JAR file are added to the Export-Package header of the bundle.

Creating an OSGi bundle from a JAR file in a workspace project

Procedure

1. Access the Copy or Move into OSGi bundle wizard. Right-click a JAR file in a project in your workspace and select **OSGi > Copy into OSGi bundle** or select **OSGi > Move into OSGi bundle**. If the JAR file is in an OSGi bundle project, you are presented with the **Move into OSGi bundle** wizard to help avoid the possibility of the JAR file being present in two different bundle projects. For other project types, you are presented with the **Copy into OSGi bundle** wizard.
2. Choose a bundle or create a new bundle to add the JAR file to. In the wizard, if you have an existing bundle in your workspace that you want to add the JAR file to, select the bundle from the drop-down menu. If you want to create a bundle for the JAR file, click **New Bundle**. If you are creating a new bundle, follow the steps of the wizard. When done, click **Finish** to return to the import wizard. The **JAR file** and **Bundle** fields are now populated.
3. Click **Next** to proceed to the packages screen of the wizard.
4. From the **Packages** list, select the packages in the JAR file that you want to be added to the Export-Package header in the bundle MANIFEST.MF file.
5. Click **Finish**. A bundle is now created based on the JAR file, and export entries for selected packages in the JAR file are added to the Export-Package header of the bundle.

Related concepts:

Using independent Java archives (JARs) with OSGi applications

Related tasks:

Including a JAR file in an OSGi application

Depending on a JAR file without including it in an OSGi application

Including a JAR file in an OSGi application

About this task

You can include a JAR file directly in your OSGi application. If you want to use packages in the JAR file in your application, and you want the packages to be included in a self-contained OSGi application, you can include the JAR file in your application. JAR files that you want to include in your application must be in OSGi bundle format. Perform the following steps to add a JAR file to your application.

Procedure

1. Create an OSGi bundle from the JAR file. If the JAR file is not already an OSGi bundle, create a bundle that is based on the JAR file. For details on how to create an OSGi bundle, see [Creating an OSGi bundle from a JAR file](#). If you are unsure whether a JAR file is already a bundle, open up the JAR file and ensure that there is a `META-INF/MANIFEST.MF` file in the JAR file that contains a `Bundle-SymbolicName` header with the name of the bundle.
2. Ensure that the bundle is in the workspace. If you created a bundle from the JAR file, the new bundle is in your workspace. If the JAR file was already a bundle but not in your workspace, import the bundle. To import the bundle, click **File > Import > OSGi > OSGi Bundle or Fragment**. Follow the steps of the wizard to import the bundle.
3. Add the bundle to your application. Double-click the `APPLICATION.MF` file in your OSGi application and select the **Overview** tab. The manifest editor opens. In the **Contained Bundles** section, click **Add**. Select the bundle that you want to include in your application. Click **OK** to add the bundle.

Parent topic: [Using independent Java archives \(JARs\) with OSGi applications](#)

Related concepts:

[Using independent Java archives \(JARs\) with OSGi applications](#)

Related tasks:

[Creating an OSGi bundle from a JAR file](#)

[Depending on a JAR file without including it in an OSGi application](#)

Depending on a JAR file without including it in an OSGi application

About this task

If you want to use independently developed JAR files with an OSGi application, but you do not want to include them directly as bundles added to the application, you can configure your workspace and server to use external JAR files. You might take this approach if you want to use JAR files that have packages, but you want to keep the JAR files separate. For instance, you might want to make your application more light-weight or keep external assets separate. JAR files that you want your application to depend on must be in OSGi bundle format. Perform the following steps to ensure that the JAR file is in bundle format and to configure your workspace and server for an application to depend on a JAR file.

Procedure

1. Create an OSGi bundle from the JAR file. If the JAR file is not already an OSGi bundle, create a bundle that is based on the JAR file. For details on how to create an OSGi bundle, see [Creating an OSGi bundle from a JAR file](#). If you are unsure whether a JAR file is already a bundle, open up the JAR file and ensure that there is a `META-INF/MANIFEST.MF` file in the JAR file that contains a `Bundle-SymbolicName` header with the name of the bundle.
2. Create a directory on your file system for the JAR file. In the following steps, you configure the location of the JAR archive file for the target platform information in your workspace and the internal bundle repository information on your server. Create a directory for the JAR file and make note of the location.
3. Export or copy the bundle to the directory you created in the previous step. If the JAR file was already in bundle format on your file system, copy it to the directory you created. To convert the JAR file to bundle format, you now have a new bundle in your workspace that you can export. To export the bundle from the workspace, right-click the bundle and select **Export > OSGi Bundle or Fragment**. In the export dialog, click **Browse** to select the location to export the bundle to. Ensure that the field **To JAR file** has the name and location that you want. Click **Finish**. The bundle is exported.
4. Edit your workspace target platform definition to include the bundle. By adding the bundle to your target platform definition, applications that depend on the bundle can compile successfully.
 - A. Access **Target Platform** preferences. Click **Window > Preferences > Plug-in Development > Target Platform**.
 - B. In the **Target definitions** section, choose the target platform that you want to compile your application against. For example, choose `WebSphere Application Server v8.0`. Click **Edit**. The **Target Content** dialog opens.
 - C. On the **Locations** tab, click **Add**. In the **Add Content** dialog, click **Directory**. Click **Next**. In the **Add Directory** dialog, browse to the directory on your file system where the bundle is located. Click **Next**. Your bundle should be in the **Plug-ins** list.
 - D. Click **Finish** to exit the **Add Directory** dialog. Click **Finish** again to exit the **Target Content** dialog. Click **OK** to exit the **Target Platform** preferences. Your workspace is now configured to compile applications that rely on packages in the bundle.
5. Add the bundle to the server. By adding the bundle to the server, you are ensuring that the server can find the packages that your OSGi application runs after it is deployed. **Note:** This step is for WebSphere® Application Server users. If you are using a different application server, see your server documentation for how to add bundles to the server.

From the **Servers** view, right-click your server and select **Administration > Run Administrative Console**. In the administrative console, browse to **Environment > OSGi bundle repositories > Internal bundle repository**. Click **New**. In the **Path to bundle** section, select **Local file system**. Click **Browse** and browse to the bundle on your file system. Click **OK** to exit the **Internal bundle repository** dialog. Click the **Save** link. The bundle is added to the internal bundle repository and the changes on the server are saved. An OSGi application that depends on the bundle can now resolve the packages on the server after the application is deployed.

Parent topic: [Using independent Java archives \(JARs\) with OSGi applications](#)

Related concepts:

[Using independent Java archives \(JARs\) with OSGi applications](#)

Related tasks:

[Creating an OSGi bundle from a JAR file](#)

[Including a JAR file in an OSGi application](#)

Extending applications by selecting composite bundles

Extend an OSGi application by selecting composite bundles in the manage extensions dialog.

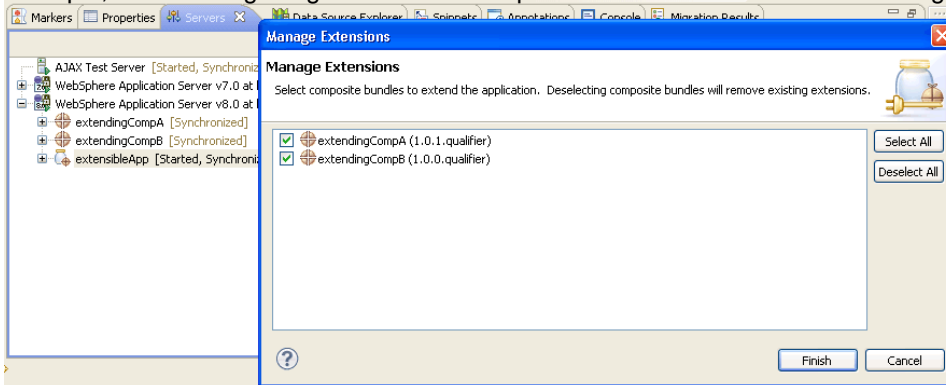
About this task

Important: Applicable edition: Full profile

You can extend the functionality of a deployed OSGi application by selecting deployed composite bundles in the manage extensions dialog.

Procedure

1. Access the servers view. If the servers view is not available click **Window > Show View > Servers**.
2. In the servers view, right-click a deployed OSGi application and select **Manage Extensions**. The manage extensions dialog is displayed and lists any composite bundles that are also deployed to the server from the same workspace. For example, the following image shows two composite bundles available for extending an application named extensibleApp.



3. Select the composite bundles that you want to extend the application with and click **Finish**.

Parent topic: [Developing OSGi applications](#)

Associating composite bundle extensions with an application

Extend an OSGi application by associating composite bundles with an application.

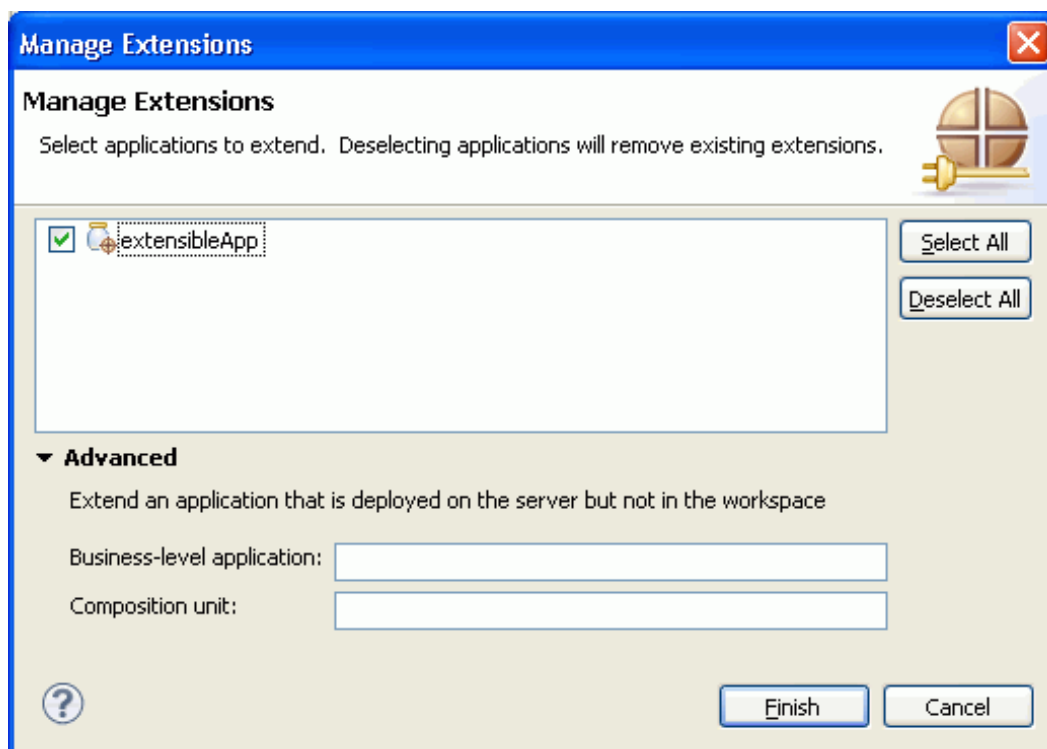
About this task

Important: Applicable edition: Full profile

You can extend the functionality of a deployed OSGi application by associating deployed composite bundles with an application.

Procedure

1. Access the servers view. If the servers view is not available click **Window > Show View > Servers**.
2. Right-click a deployed composite bundle and select **Manage Extensions**. The manage extensions dialog is displayed and lists OSGi applications that are also deployed to the server from the same workspace.
3. Select the applications that you want to extend with the composite bundle. For example, the following image shows an application named extensibleApp is available for extending.



4. (Optional) Advanced option for extending applications on the server. You can extend OSGi applications that are deployed to the server that are not available in the server view in your workspace. To extend these OSGi applications, in the **Advanced** section, enter the **Business-level application** and **Deployed asset** information for the deployed application:

- Business-level application

- The name of the deployed business-level application that you want to extend. In the WebSphere® Application Server administrative console, you can find business-level applications under **Applications > Application types > Business-level applications**.

- Deployed asset

- The name of the deployed asset for the business-level application. To find a deployed asset name, in the administrative console, click the link for the business-level application that you want to extend. In the page that is displayed for the application, the deployed asset name is listed under the section **Deployed assets**.

5. Click **Finish** when done with the manage extensions dialog. If there are more applications on the server that you want to extend, access the dialog again and add another application.

Parent topic: [Developing OSGi applications](#)

Exposing EJBs as OSGi services

You can include EJBs in OSGi bundles and expose EJBs as OSGi services.

About this task

Important: Applicable edition: Full profile

Procedure

1. Configure an OSGi bundle for EJB support. There are two primary methods to configure an OSGi bundle for EJB support:

- Convert an EJB project to an OSGi bundle project

- To convert an EJB project to an OSGi bundle project, right-click the project and select **Configure > Convert to OSGi Bundle Project**. After you convert the EJB project, the OSGi bundle facet is added to the project and a bundle manifest file is created.

- Add EJB support when you create an OSGi bundle

- You can add EJB support when you create an OSGi bundle. To add EJB support, click **File > New > OSGi Bundle Project**. In the bundle project wizard, select **Add EJB support** and select the EJB level that you want to use.

With both of these methods, a header called Export-EJB is added to the bundle manifest file. EJBs that you want to expose as services can be listed next to this header.

2. Manage EJB exports. To control the EJBs that you want to expose as OSGi services, right-click the OSGi bundle project and select **OSGi > Expose EJBs as OSGi services**. In the manage EJB exports dialog that opens, select the EJBs that you want to expose as services.

3. Understand the manifest. Double-click the Manifest file in a bundle project to open it in the manifest editor. Select the **MANIFEST.MF** tab to view the manifest source. OSGi bundle projects that have EJB support have a header called Export-EJB. EJBs are entered as a comma-separated list. This example shows two EJBs exported as services:

```
EJB: EJB1, EJB2
```

Note: There are two special cases of the Export-EJB header to be aware of:

- NONE

- If you specify `NONE` as an entry for the Export-EJB header, no EJBs are exposed as services. If you specify `NONE`, but then also add an EJB to the list, a warning is displayed by the tools.

- BLANK

- If you have an Export-EJB header in the manifest but there are no entries, by default all EJBs in the project are exposed as services.

4. Add and delete EJBs automatically. By default, when you add or delete an EJB to an OSGi bundle project with EJB support, the EJB entry is automatically added or removed from the Export-EJB header in the manifest file.

Parent topic: [Developing OSGi applications](#)

Related information:

[OSGi EJB tutorial](#)

Configuring OSGi bundle repositories

About this task

Important: Applicable edition: Full profile

OSGi bundle repositories are external collections of bundles that are described in a repository XML file. If your OSGi application requires bundles provided in external repositories to compile correctly, you can configure your workspace to load the bundles. You can also deploy the bundle repository description to your server so that the server can access the required bundles after your application is deployed.

The following steps describe how you can access the OSGi bundle repositories view and learn the options for configuring bundle repositories.

Procedure

1. Access the bundle repositories view. Click **Window > Show View > Other > OSGi > OSGi Bundle Repositories**. The OSGi bundle repositories view is opened in your workspace.
2. Add a repository. Right-click the **OSGi Bundle Repositories** icon. Select **Add Repository**. The **New OSGi Bundle Repository** dialog is displayed. Enter the information that is required for the repository and click **OK** when finished:
 - **Repository URL**
 - The URL for the repository description file. The repository description file can be available on a network over HTTP or it can be on your local file system. If the file is available over HTTP, enter the URL and click **Load**. If the file is local, click **Local** and browse to the file. **Note:** For files available over HTTP, the **OK** button in the dialog will be disabled until the file is loaded successfully.
 - **Target Platform**
 - The target platform that you want to compile your application against. For example, `webSphere Application Server v8.0`.
3. Select the bundles that you want to load from the repository that you added. You can select multiple bundles by holding down the Ctrl key while you select.
4. Load bundles. Right-click the selected bundles and click **Load Bundle**. Loading a bundle caches a copy of the bundle in the `.metadata` folder of your workspace so that projects that require the bundle compile correctly. **Note:** The bundle is not visible in the projects view of your workspace.

Note: Finding bundles to load by service If a bundle has a service that uses an API in another bundle and you want to load the bundle that contains the API, you can quickly identify and load that bundle. Expand the bundle that contains the service. Right-click the service and select **Load Service API...**. The **Load service API bundle** dialog displays the bundles that contain the APIs used by the service. Select the bundles that you want to load, and click **OK**.
5. Deploy a bundle repository description to the server. If you want your server to be able to access bundles in a bundle repository that are required by your application, deploy the bundle repository description to the server. Right-click the bundle repository that you added and select **Deploy to Server**.
6. Explore other actions in the bundle repository view. Other actions available in the bundle repository view are **Associate Target**, **Refresh**, and **Delete**. To access these actions, right-click an existing bundle repository. Descriptions for these actions are in the following list:
 - **Associate Target**
 - Change the target platform that you want to compile your application against. For example, `WebSphere Application Server v8.0`.
 - **Refresh**

- If you changed the bundles on a repository, click **Refresh** to load the latest versions of the bundles.

- Delete

- Click **Delete** to remove a repository from the bundle repository view.

Parent topic: [Developing OSGi applications](#)

OSGi application samples and tutorials

To view the sample and tutorials in this product, click **Help > Help Contents** and expand the Samples and Tutorials sections. Learn about different aspects of OSGi application development from the following samples and tutorials:

-  **Sample: OSGi Hello World**

- This sample OSGi application contains a servlet that demonstrates the use of an activator.

-  **Sample: OSGi Counter Service**

- This sample OSGi application consists of an OSGi web bundle that contains a servlet that accesses a service that is provided in another bundle project. This sample is an introduction to using OSGi application development tools.

-  **Sample: OSGi Blog**

- This sample OSGi application demonstrates how to structure the API and implementation code into separate bundles. The OSGi application consists of an OSGi web bundle that contains servlets that access a JPA service that is provided in another bundle.

-  **Sample: EJB temperature converter**

- This OSGi sample demonstrates an EJB configured as an OSGi bundle and exposed as a service.

-  **Tutorial: Develop a simple OSGi application**

- This tutorial demonstrates how to create an OSGi application and run it on WebSphere Application Server. The OSGi application consists of an OSGi web bundle that contains a servlet that accesses a service that is provided in another bundle project. This tutorial is an introduction to using OSGi application development tools.

-  **Tutorial: OSGi EJB service**

- This tutorial demonstrates how to create an OSGi application that exposes an EJB as a service. It demonstrates how to create OSGi bundles with EJB support, use the OSGi tools to manage EJB exports, and create a servlet that accesses the EJB as an OSGi service.

Parent topic: [Developing OSGi applications](#)

Developing Apache Maven projects

Apache Maven is a framework that can help with build automation.

About this task

Important: Applicable editions: Liberty profile, full profile

Maven assists developers and build coordinators in unifying a wide range of build requirements and standards into one system that is flexible enough for customization. The framework provides a common API that can be learned once and deployed company wide. Instead of writing custom Ant scripts, developers can build Maven commands that can be duplicated and reused. Without Maven, bootstrapping a new build process according to established company standards can be cumbersome, and resources cannot be shared across small and large organizations. Maven solved many build problems by standardizing dependency management, artifact repositories, project archetype structures and schemes, adapters that connect to external source code management (SCM), test systems, and build technologies.

- [Maven overview](#)

Several key concepts are important to understand in the Maven framework.

- [Setting up your environment](#)

Before you begin Maven development, you must set up your environment.

- [Defining Java EE module dependencies](#)

For non-Maven Java™ EE projects, the Deployment Assembly property page is used to define dependencies to other projects, libraries, and utilities. The page is also used to describe the assembly of the archive at run time, including assembling modules within an EAR. However, Maven projects define dependencies within the `pom.xml` file and generate and derive dependencies that are used by the project and publishing mechanisms. All dependencies must be defined within the `pom.xml` file. Dependencies can be defined in source under the `<dependencies>` tag or on the POM editor dependencies tab.

Related information:

[↗ Apache Maven](#)

Maven overview

Several key concepts are important to understand in the Maven framework.

Important: Applicable editions: Liberty profile, full profile

- Project Object Model (POM)

- Each Maven project provides a `pom.xml` file that captures dependencies, project structure properties, build phase tasks and behavior. Most of the POM properties have defaults that result in a compact but powerful mechanism for describing project build behavior.

- Build phases and the build lifecycle

- A build lifecycle consists of several phases. When a phase command is given, Maven runs every phase in the sequence up to and including the defined phase. After the `pom.xml` file is defined, the Maven tools prioritize specific build phases and react to phases from validate, code generation, resource assembly, and compilation. A build lifecycle consists of the following phases:
 - validate
 - compile
 - test
 - package
 - integration-test
 - verify
 - install
 - deploy

More information on the build lifecycle can be found at <http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

- Goal

- A goal represents a specific task that is finer than a build phase and that contributes to the building and managing of a project.

- Packaging

- Setting the packaging assigns a set of default goals. Examples of valid packaging values include jar, war, ear, and pom.

- Maven plug-in

- A plug-in describes a set of goals that are tied to a specific packaging scheme or process.

- Mojo

- A specific task that is implemented inside a plug-in. For example, a Java™ class implementation for deploying to your preferred runtime environment.

- Archetype

- Archetypes are used as project templates for setting up new projects. These templates make it easy to enable standards within your organization by defining packaging goals, plug-in settings, and predefined dependencies to standard libraries.

- Maven repositories

- Repositories are used to store build artifacts and dependencies of varying types. For example, archetypes, plug-ins, and JAR files, among others. Local repositories are populated lazily as needed from remote repositories for build purposes.

- Tools and projects overview

There are several tools that are provided for Maven development that you can use to define Java EE module dependencies. You can also use POM entries to define project structure.

Parent topic: [Developing Apache Maven projects](#)

Tools and projects overview

There are several tools that are provided for Maven development that you can use to define Java™ EE module dependencies. You can also use POM entries to define project structure.

Important: Applicable editions: Liberty profile, full profile

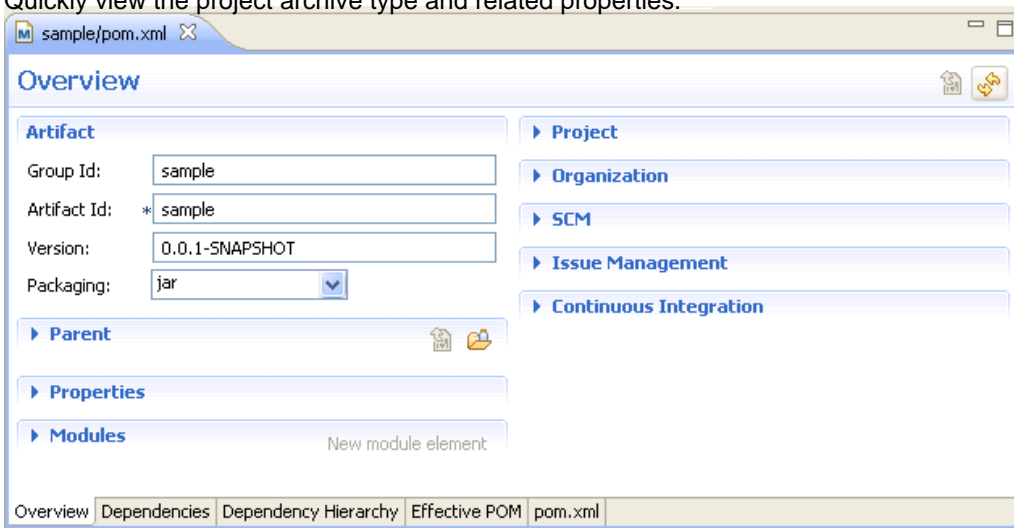
- [POM editor overview](#)
- [Maven Repositories view](#)
- [POM entries that affect the development workbench project structure](#)

POM editor overview

To open the POM editor, double-click a `pom.xml` file in a Maven enabled project. This multi-tabbed, form-based editor simplifies editing of the Maven object model.

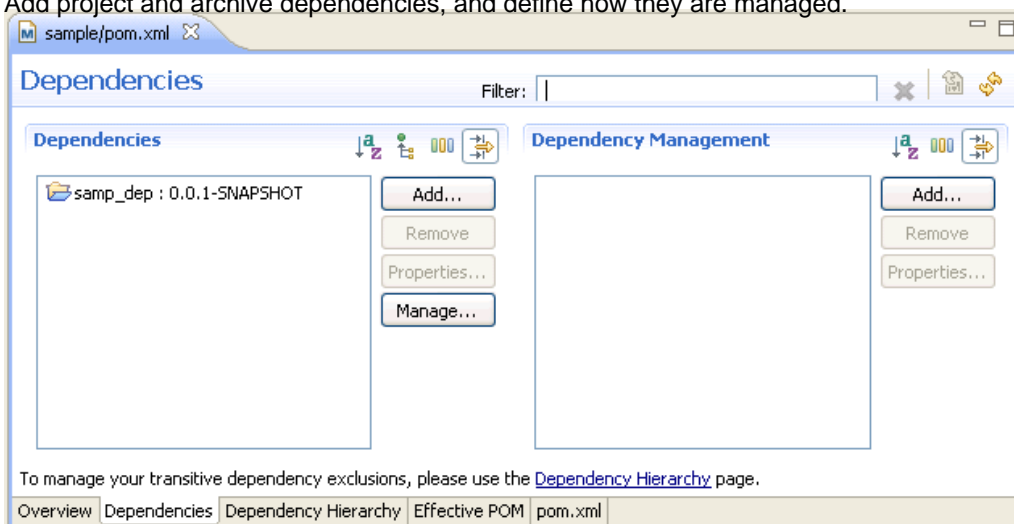
- Overview page

- Quickly view the project archive type and related properties.



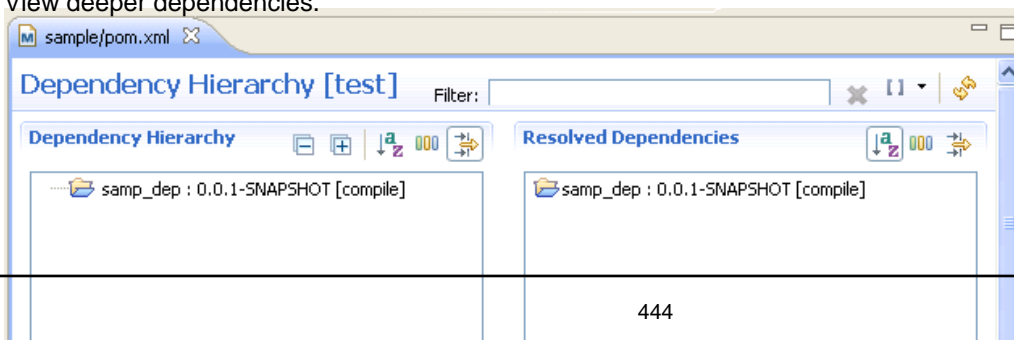
- Dependencies page

- Add project and archive dependencies, and define how they are managed.



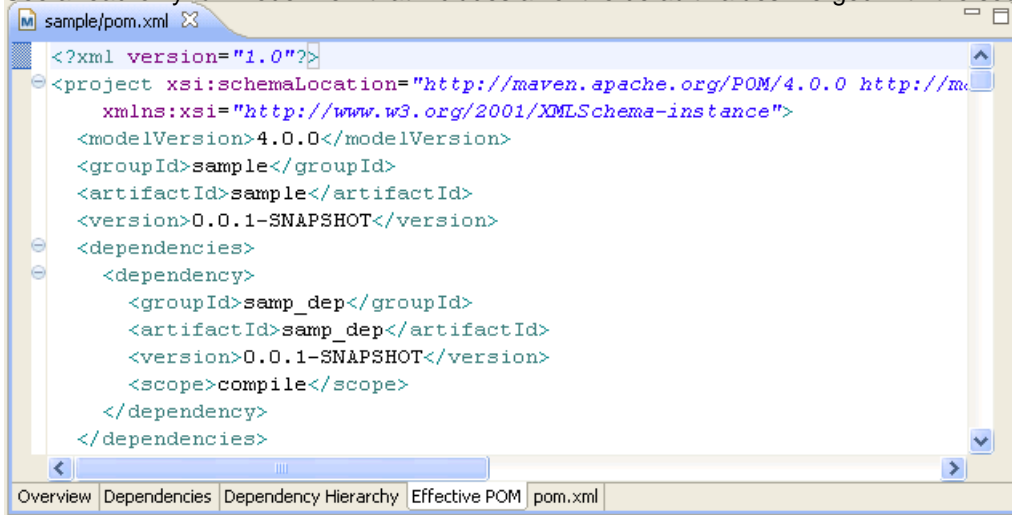
- Dependency Hierarchy

- View deeper dependencies.



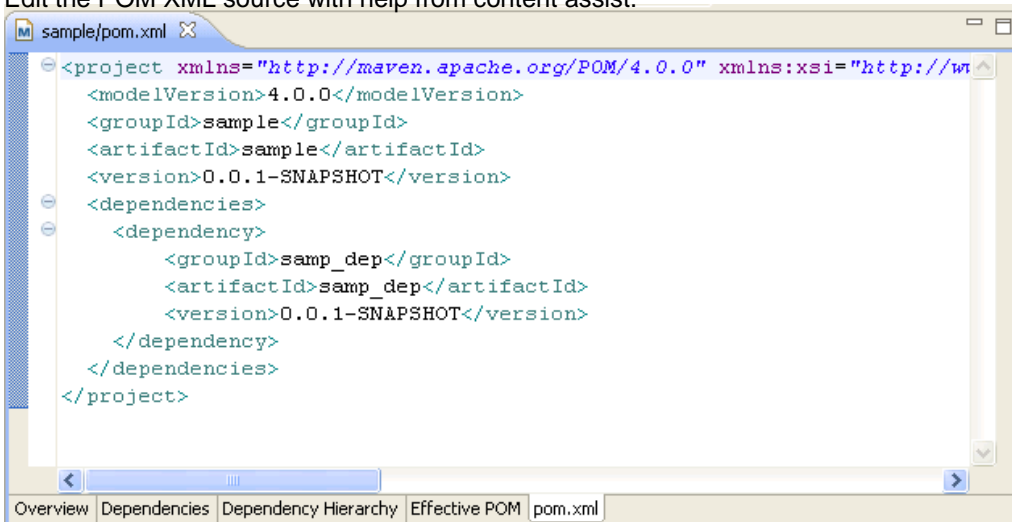
- Effective POM

- See a read-only full-model view that includes all of the default values merged with the source pom.xml.



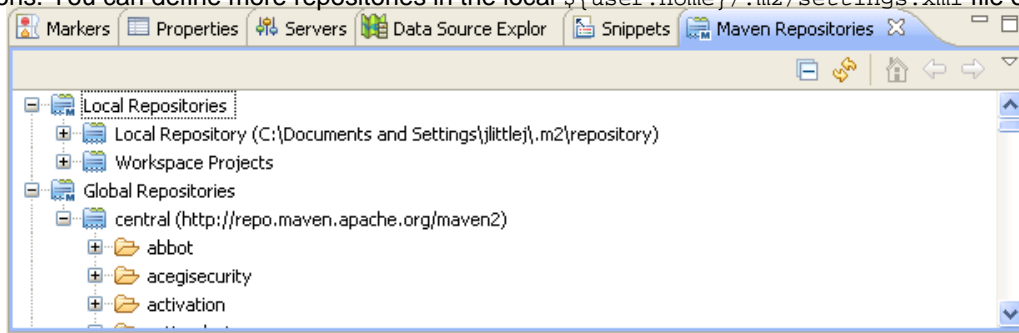
- pom.xml (source view)

- Edit the POM XML source with help from content assist.



Maven Repositories view

In the Maven Repository viewer, you can drill down and search local and remote repositories that are used for building your applications. You can define more repositories in the local `${user.home}/.m2/settings.xml` file or in the pom.xml file of



a project.

POM entries that affect the development workbench project structure

The following table shows several scenarios and attributes that are described in the Maven pom.xml and their affect on development workbench project settings.

Scenario	POM attribute	Project structure change
Set the source folder location.	<code><build> <sourceDirectory></sourceDirectory></code>	Java build path settings are changed to this source folder.
Set the output folder location.	<code><build> <outputDirectory></outputDirectory></code>	Java build path settings are changed to this output folder.
Define the archive name for EAR or WAR files.	<code><build> <finalName></finalName></code>	Changes the deploy name for the project and creates the archive name upon export and deployment.
Add dependencies using the MANIFEST file in an EJB project.	<code><archive> <manifest> <addClasspath>true</addClasspath></code>	Generated MANIFEST.MF file includes dependencies.
Use the MANIFEST file in the src folder.	<code><plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-jar-plugin</artifactId> <configuration> <archive> <manifestFile>src/main/java/META-INF/MANIFEST.MF</manifestFile> </archive></code>	Use the existing MANIFEST.MF file in the deployed JAR file.
Create a WAR module.	<code><packaging>war</packaging> + <plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-war-plugin</artifactId> ...</code>	The dynamic web facet is specified and WAR plugin settings are read for detailed changes.
Do not generate an EAR deployment descriptor XML.	<code><generateApplicationXml>false</generateApplicationXml></code>	The Application.xml file is not generated in the output location.

Parent topic: [Maven overview](#)

Setting up your environment

Before you begin Maven development, you must set up your environment.

About this task

Important: Applicable editions: Liberty profile, full profile

1. [Enable Maven repository indexing.](#)
2. [Set up the IBM® Maven repository to access IBM provided archetypes and POM files.](#)
3. [Configuring dependency POM files that emulate the classpath of specific WebSphere runtime environments.](#)

- [Downloading and enabling the Maven repository index](#)

Before you work with Maven, you must download and enable the Maven repository index.

- [Setting up the IBM Maven repository](#)

The IBM Maven repository provides archetypes and POM files that you can use to develop Maven applications.

- [Configuring dependency POM files that emulate the classpath of specific WebSphere runtime environments](#)

8.5.5.2 If you plan to use target Project Object Model (POM) files that emulate the classpath of WebSphere runtime environments, you can configure your maven project dependencies and perform the steps required to use those dependencies.

- [Installing the server APIs into the Maven repository](#)

Parent topic: [Developing Apache Maven projects](#)

Downloading and enabling the Maven repository index

Before you work with Maven, you must download and enable the Maven repository index.

Before you begin

Important: Applicable editions: Liberty profile, full profile

- You must be connected to the Internet.
- You must have at least 400 MB of space available on the drive that contains your workspace.

About this task

When you create a new workspace, Maven repository indexing is turned off by default.

Procedure

Click **Window** > **Preferences** > **Maven** and select **Download repository index updates on startup**. **Note:** Downloading and enabling the Maven repository index increases the size of your workspace by at least 400 MB. Therefore, select this option only if you plan to develop Maven applications.

Parent topic: [Setting up your environment](#)

Setting up the IBM Maven repository

The IBM® Maven repository provides archetypes and POM files that you can use to develop Maven applications.

About this task

Important: Applicable editions: Liberty profile, full profile

The IBM Maven repository supports development of Maven applications for the following runtime environments:

- WebSphere® Application Server V8.5.5 and V8.5.0.2.
- WebSphere Application Server Liberty Profile V8.5.5 and V8.5.0.2.
- WebSphere Portal V8.0.

Procedure

1. Click **Window > Preferences > Java EE > Maven > Maven Repository Initialization**. If the `settings.xml` file does not exist, the page displays the following text: `The settings.xml file does not exist. To create this file and configure the IBM Maven repository, click Configure.` This action configures the IBM Maven repository in the `settings.xml` file and registers the URL of the archetype catalog.**Note:** If the `settings.xml` file does not exist, a new file is created. If the `settings.xml` file does exist, content is updated and not overwritten.
2. The location of the `settings.xml` file is determined by the value that is set in the **Window > Preferences > Maven > User Settings** page. To change the file location, click the **User settings file** link to go directly to the preference page.
3. Click **OK** to exit the preference dialog. The Updating indexes job displays in the Progress view.

Parent topic: [Setting up your environment](#)

Configuring dependency POM files that emulate the classpath of specific WebSphere runtime environments

8.5.5.2 If you plan to use target Project Object Model (POM) files that emulate the classpath of WebSphere runtime environments, you can configure your maven project dependencies and perform the steps required to use those dependencies.

About this task

Important: Applicable editions: Liberty profile, full profile

8.5.5.2 Target POM files are available that represent the server runtime environment for compilation purposes. Target POM files are available for the following runtimes, and you can add a corresponding fragment as a dependency:

8.5.5.2

- WebSphere Application Server Version 8.5.5<dependency>

```
<groupId>com.ibm.tools.target</groupId>
<artifactId>was</artifactId>
<version>8.5.5</version>
<type>pom</type>
<scope>provided</scope>
```

</dependency>

- WebSphere Application Server Version 8.5.0.2<dependency>

```
<groupId>com.ibm.tools.target</groupId>
<artifactId>was</artifactId>
<version>8.5.0</version>
<type>pom</type>
<scope>provided</scope>
```

</dependency>

- [Liberty profile] WebSphere Application Server Liberty Profile Version 8.5.x (Includes all 8.5 maintenance releases)

<dependency>

```
<groupId>com.ibm.tools.target</groupId>
<artifactId>was-liberty</artifactId>
8.5.5.4 <version>LATEST</version>
<type>pom</type>
<scope>provided</scope>
```

</dependency>

- [Liberty profile] Optional dependency for compilation against 3rd party implementation libraries, such as Open JPA, Wink, and Jackson<dependency>

```
<groupId>com.ibm.tools.target</groupId>
<artifactId>was-liberty-impl</artifactId>
8.5.5.4 <version>LATEST</version>
<type>pom</type>
<scope>provided</scope>
```

</dependency>

- WebSphere Portal Version 8.0<dependency>

```
<groupId>com.ibm.tools.target</groupId>
<artifactId>portal</artifactId>
<version>8.0.0</version>
```

```

<type>pom</type>
<scope>provided</scope>
</dependency>

```

8.5.5.2 The dependencies for WebSphere Application Server Liberty Profile Version 8.5.x target POM do not require any further configuration. For all remaining POM targets, you are required to use launch configuration scripts that copy runtime libraries to the local Maven repository.

The scripts are in the `com.ibm.etools.maven.javaee.core` plug-in, which is located under the `plugins` directory in the product installation.

Procedure

- To import the launch configuration script for the target runtime environment, click **File > Import > Run/Debug > Launch Configurations**.
- Click **Browse** to locate the directory where the scripts are, and then select them.
 - For WebSphere® Application Server, the scripts are located in: `<IBMSharedFolder>\plugins\com.ibm.etools.maven.javaee.core_x\resources\RunConfigurations\WAS855PluginsInstaller.launch`
Note: This script can also be used for WebSphere Application Server V8.5.0.2
 - For WebSphere Portal V8.0: `<IBMSharedFolder>\plugins\com.ibm.etools.maven.javaee.core_x\resources\RunConfigurations\WAS8PluginsInstaller.launch`

Note: X in `com.ibm.etools.maven.javaee.core_x` is the version of the plug-in.
- Click **Finish**.
- To modify the launch configuration, click **Run > Run configurations**. Expand **Maven Build** to see the imported configuration, and then click the launch configuration to open it.
- In the Base directory field, specify the location where the `pom.xml` can be found:
 - WebSphere Application Server V8.5.5: `<IBMSharedFolder>\plugins\com.ibm.etools.maven.javaee.core\resources\scripts\WAS 8.5.5 Plugins Installer`
 - WebSphere Application Server V8.5.0.2: `<IBMSharedFolder>\plugins\com.ibm.etools.maven.javaee.core\resources\scripts\WAS 8.5.0 Plugins Installer`
 - WebSphere Portal V8.0: `<IBMSharedFolder>\plugins\com.ibm.etools.maven.javaee.core\resources\scripts\WAS 8 Plugins Installer`
- Specify a value for the **serverInstallationFolder** parameter.
 - Select the parameter and click **Edit**.
 - In the **Value** field, specify the directory where the server is installed and then click **OK**:
 - WebSphere Application Server: `<server_installation_directory>\AppServer`
- Click **Run** to run the launch configuration.
- Open the Maven Repositories view by clicking **Window > Show View > Other > Maven > Maven repositories**.
- In the Maven repositories view, expand **Local Repositories**. Right click the node **Local Repository (C:\Users\...\m2\repository)** and select **Rebuild Index**.
- Expand the node **Local Repository (C:\Users\...\m2\repository)** and verify the applicable jar file. In this case, `com.ibm.websphere.j2ee.j2ee.1.0.0` is listed. Your local repository should look like this: `Local Repositories`

```

> Local repository (C:\Users\...\m2\repository)
> com\
> ibm\
> websphere\

```

```

> j2ee\
  (jar icon) j2ee - jar
> [M] j2ee : x.y.z

```

11. Optional: To verify that the jar files were installed correctly, create the project using the archetype `com.ibm.tools.archetype.webapp.jee6-was`. For more information, see [Creating Maven projects](#).

12. Optional: Check the contents of the Maven Dependencies classpath container of the project. In the Enterprise Explorer view, it looks like this: `M(maven icon)J(java icon) project folder {YourProjectName}`

```

> (web project icon) {YourProjectName}
  > Java Resources
    > src/main/java
    > Libraries
      > JRE System Library [JavaSE-1.6]
      > Maven Dependencies
        - com.ibm.ws.wccm-1.0.0-jar - {respository path to jar}
        - com.ibm.ws.emf-1.0.0-jar - {respository path to jar}
        - j2ee-1.0.0-jar - {respository path to jar}

```

What to do next

Note: You can alternatively run the scripts from the Maven command line by specifying the `command:mvn install`

```
-f <IBMSharedFolder>
```

```
\plugins\com.ibm.etools.maven.javaee.core\resources\scripts\<plugin_installer_folder>\pom.xml"
```

```
-DserverInstallationFolder="<server_installation_directory>" 8.5.5.2 For example: mvn install -f
```

```
"<IBMSharedFolder>\plugins\com.ibm.etools.maven.javaee.core\resources\scripts\WAS
```

```
8.5.5 Plugins Installer\pom.xml" -DserverInstallationFolder="C:\Program
```

```
Files\IBM\WebSphere\Appserver"
```

Parent topic: [Setting up your environment](#)

Installing the server APIs into the Maven repository

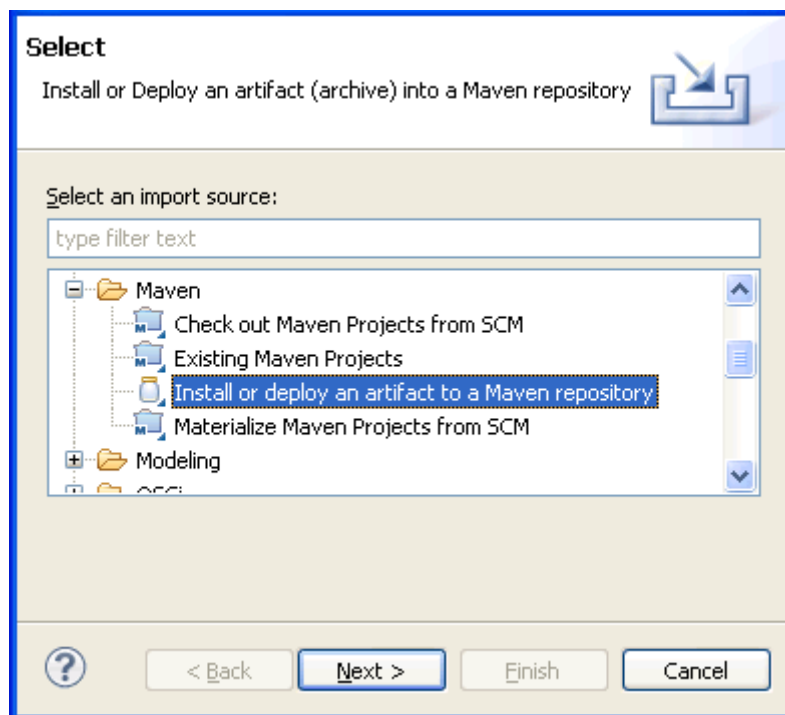
About this task

Important: Applicable edition: Full profile

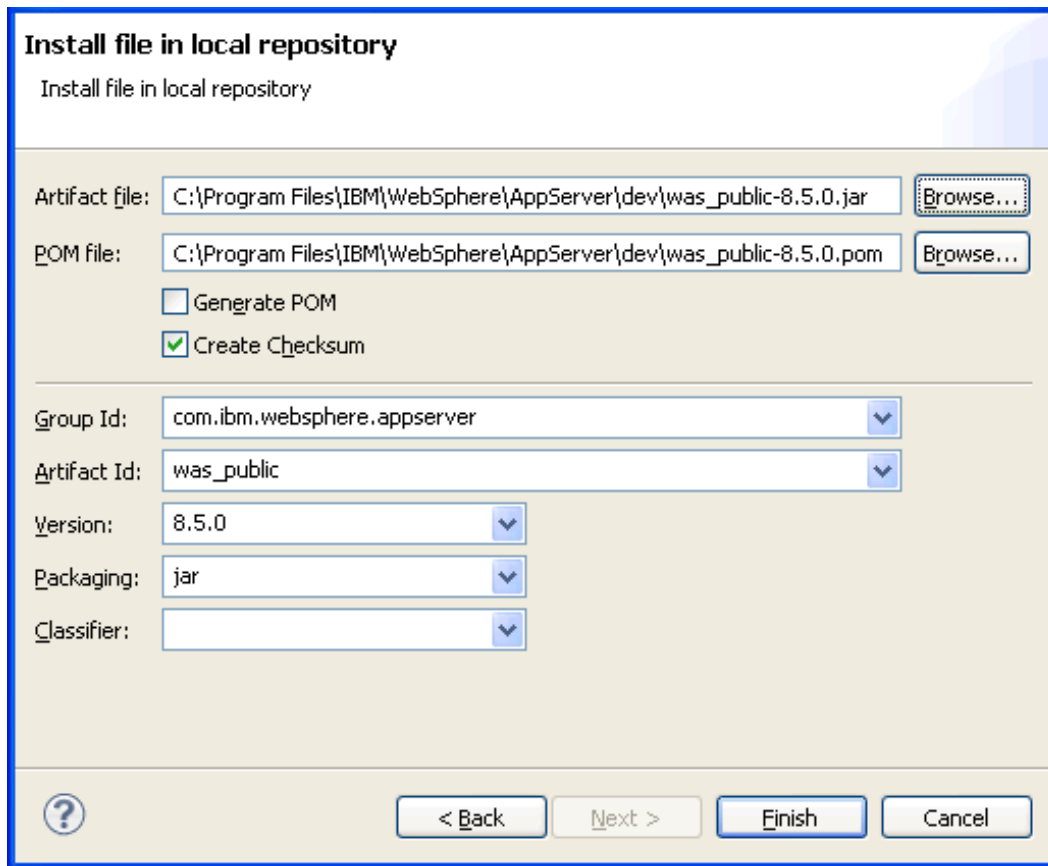
WebSphere® Application Server V8.0 and V8.5 provide a Maven module with the APIs that are required to compile your projects. If you install this module into your Maven repository, you do not need to manually add the module JAR file to your class path. **Note:** This information assumes that you have WebSphere Application Server installed on a system that you have access to.

Procedure

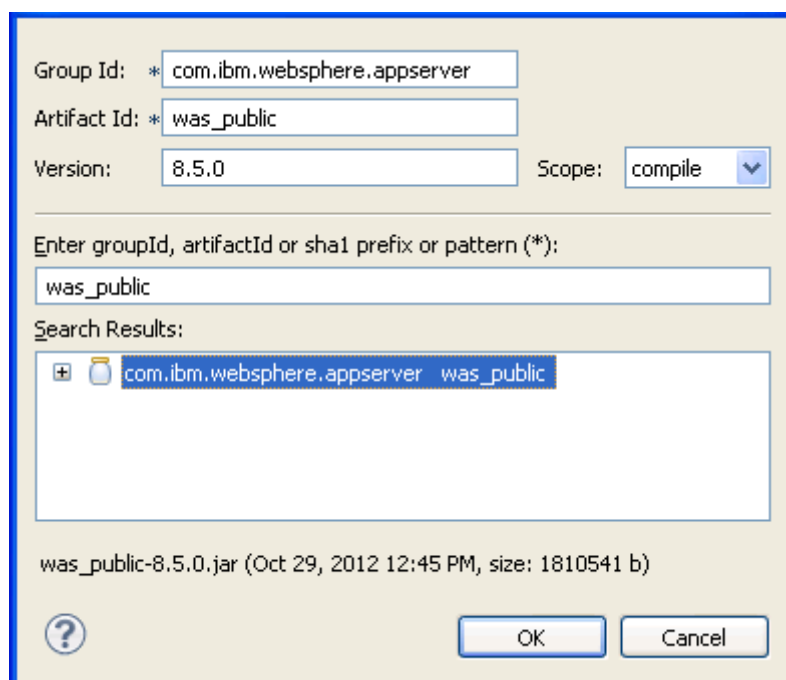
1. Locate the module JAR file, `was_public.jar`. This JAR file contains the APIs and is in the `dev` directory of your WebSphere Application Server installation. For example, if your application server is installed into `C:\Program Files\IBM\WebSphere\AppServer`, the JAR file is located at `C:\Program Files\IBM\WebSphere\AppServer\dev\was_public.jar`.
2. Rename the JAR file to match the server version. If you are using version 8.0.0, rename the file to `was_public-8.0.0.jar`. If you are using version 8.5.0, rename the file to `was_public-8.5.0.jar`.
3. Access the install artifact dialog. Click **File > Import > Maven > Install or deploy an artifact to a Maven repository**. Click **Next**.



4. Install the file to your local repository. Click **Browse** next to the **Artifact file** field and browse to the JAR file that you renamed in the previous steps. When you populate the **Artifact file** field, the **POM file** field is automatically updated with the name of the POM file that is also in the server `dev` directory. The rest of the required fields are also updated.



5. Click **Finish** to install the file to your local repository. **Note:** The module needs to be installed to a local repository only once per machine. To find the location of the repository, click **Window > Preferences > Maven > User Settings** and note the value in the **Local Repository** field.
6. So that a project can compile against the module, add a dependency to the module. If you have an existing Maven project, right-click the project and select **Maven > Add Dependency**. In the search filter field, enter `was_public`. The search results display the module. Select the module; the **Group Id** and **Artifact Id** fields are updated. Set **Scope** to `compile`. Click **OK** to complete the dialog. A project dependency to the `was_public` module is added and it is not necessary to add the module JAR file to your class path.



Parent topic: [Setting up your environment](#)

Related tasks:

[Creating Maven projects](#)

Creating Maven projects

About this task

Important: Applicable editions: Liberty profile, full profile

You can create simple projects or projects that are based on archetypes.

- [Creating a simple Maven project](#)
- [Creating a Maven project that is based on archetypes](#)

Parent topic: [Developing Apache Maven projects](#)

Creating a simple Maven project

Procedure

1. Select **File > New > Project > Maven > Maven Project**. Click **Next**.
2. Select **Create a simple project (skip archetype selection)**. Click **Next**.
3. Enter or select values for the following fields:
 - **Group Id**
 - **Artifact Id**
 - **Version**
 - **Packaging**
 - **Name** (optional)
 - **Description** (optional)

Note: The values for **Group Id**, **Artifact Id**, and **Version** are known as Maven coordinates. For more information about Maven coordinates, see [Maven Coordinates](#).
4. Enter parent project values. If the project is a child project of another Maven project, in the **Parent Project** section, enter the **Group Id**, **Artifact Id**, and **Version** of the parent project.
5. Click **Finish**. The project is created.

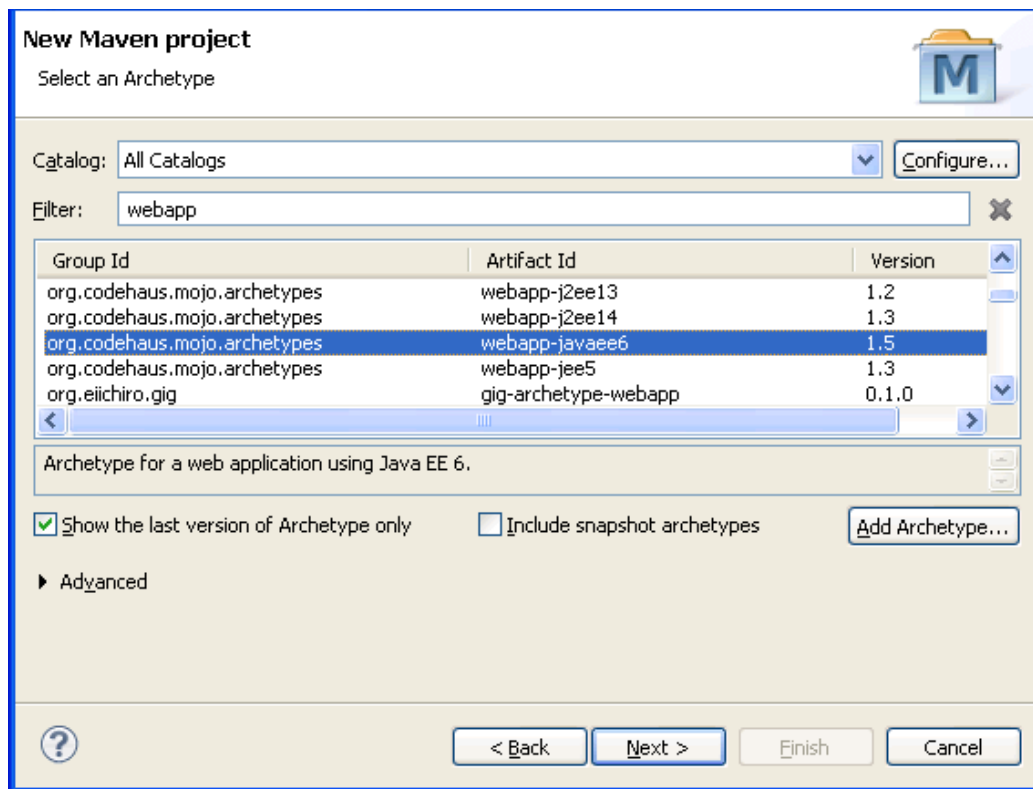
Creating a Maven project that is based on archetypes

About this task

For more information about archetypes, see [Introduction to archetypes](#).

Procedure

1. Select **File > New > Project > Maven > Maven Project**. Click **Next**. The Maven project wizard opens.
2. On the first page of the wizard, click **Next** to proceed to the **Select an Archetype** page.
3. Select the archetype that you want to use. You can select a **Catalog** or enter a **Filter** to reduce the number of archetypes that are displayed. This example in this image shows a search of all catalogs, a filter for `webapp`, and a selection of the archetype with artifact ID `webapp-javaee6`.



Click **Next**.

4. Enter or select values for the following fields:

- **Group Id**
- **Artifact Id**
- **Version**
- **Package**

Note: The values for **Group Id**, **Artifact Id**, and **Version** are known as Maven coordinates. For more information about Maven coordinates, see [Maven Coordinates](#).

5. Click **Finish**. The project is created.

Importing projects

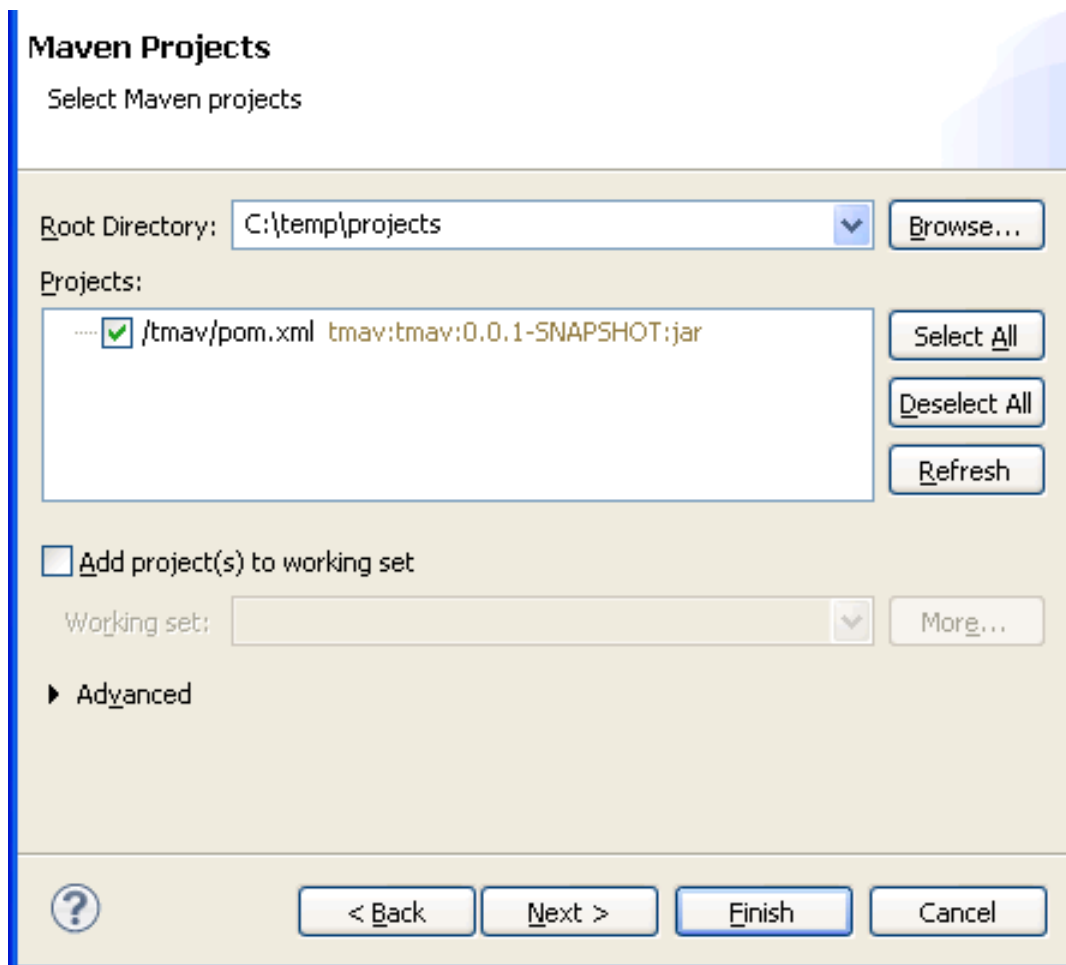
You can import Maven projects that were developed outside of the development workbench.

About this task

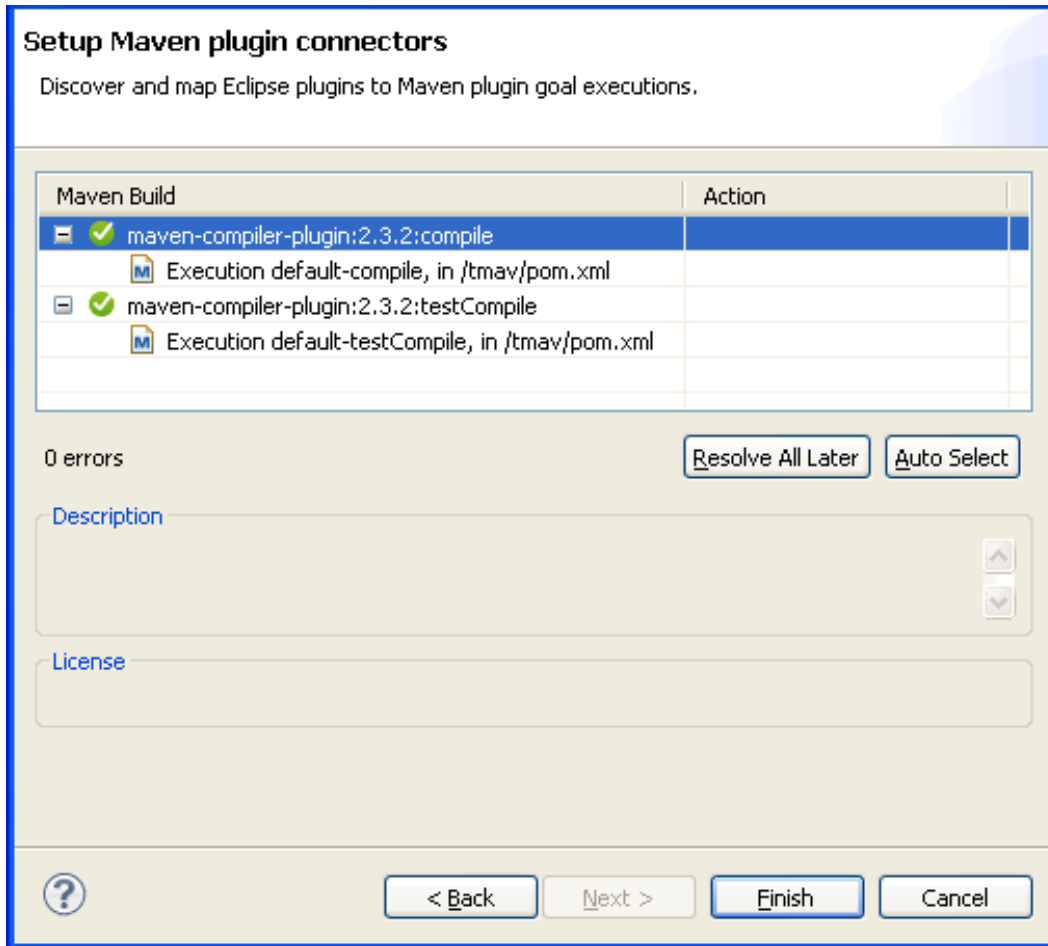
Important: Applicable editions: Liberty profile, full profile

Procedure

1. Open the import wizard. Select **File > Import > Maven > Existing Maven Projects**. Click **Next**.
2. Browse to the project. Click **Browse** and go to the **Root Directory** that contains your projects. If you have a project with multiple modules, select the directory that contains the `pom.xml` file for the top-level module. The wizard scans all the subdirectories for more projects. After scanning, all projects to be imported are listed. Click **Next**.



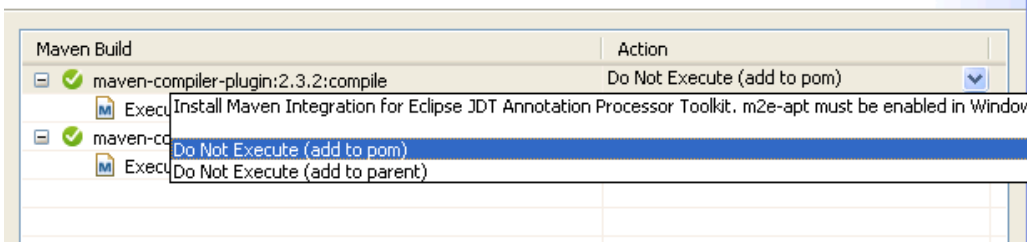
3. Set up Maven plug-in connectors. The **Setup Maven plugin connectors** page list all the Maven plug-in goals in the projects and the plug-ins that process those goals. If you see only green check marks, all of the necessary plug-ins are installed.



If you see a red mark, you must either install an additional connector or specify in the POM not to execute that goal. These actions can be specified in the **Action** column. To choose an action, select a plug-in, then click in the action column until the drop-down arrow is displayed. Click the drop-down arrow to select an action.

Setup Maven plugin connectors

Discover and map Eclipse plugins to Maven plugin goal executions.



4. Click **Finish**. A project is created for each of the Maven projects that you import.

Parent topic: [Developing Apache Maven projects](#)

Converting existing projects to Maven

You can convert existing Java™ EE projects to Maven.

Before you begin

Important: Applicable editions: Liberty profile, full profile

Back up your workspace so that you can return to the original project state if necessary.

About this task

The suggested order of tasks for converting projects is as follows:

1. [Set recommended preferences.](#)
2. [Convert non-EAR projects.](#)
3. [Convert EAR projects.](#)
 - A. [Create a content folder.](#)
 - B. [Convert the EAR project.](#)
 - C. [Add modules to an EAR.](#)
 - D. [Add dependencies to other modules.](#)
 - E. [Add libraries to the EAR library directory.](#)

Parent topic: [Developing Apache Maven projects](#)

Setting recommended preferences

About this task

To use your workspace for Maven projects, first set the recommended preferences.

Procedure

1. Access the Maven project settings preferences. Click **Windows > Preferences > Java EE > Maven > Maven Project Settings.**
2. Click **Set all Maven values.** The preferences that represent the best practices for Maven configuration are set.
3. Click **OK** to complete setting the preferences.

Converting web, EJB, connector, application client, utility, web fragment and ejb client projects

About this task

When you convert a project, keep in mind the dependencies that you have to other artifacts and how the artifacts are accessed by the project. In non-Maven projects, this information is specified in project metadata and the `MANIFEST.MF` files directly. However, in Maven projects, dependencies are specified in the `pom.xml` file. For example, the artifacts can be accessed through the `MANIFEST.MF` file or deployed in library directory of the EAR that contains the project. For web projects, artifacts can be deployed in the `WEB-INF/lib` directory.

First, use the deployment assembly page to gather the dependencies information.

1. Right-click the project.

2. Select **Properties > Deployment Assembly**.
3. Click the **Manifest Entries** tab. Make note all of the entries that are referenced by the project.
4. For web projects, also click the **Deployment Assembly** tab. Make note all of the projects and JAR files with a deploy path that begins with `WEB-INF/lib`.

Note: If the EAR project that contains your projects has EAR facet version 5 or 6, then your project might be referencing classes from projects or JAR files in the library directory of the EAR file. Make note of these references as well.

After you gather the required information, you can convert the project to Maven.

Procedure

1. Right-click the project and select **Configure > Convert to Maven Project**.
2. Complete the Maven POM dialog. Enter a **Group Id**, **Artifact Id**, and **Version** or accept the defaults. Optionally, enter a **Name** and **Description**. For **Packaging**, choose according to the following project types:
 - **Web projects**
 - `war`
 - **EJB projects**
 - `ejb`
 - **Connector, Utility, web fragment, and EJB client projects**
 - `jar`
 - **Application client projects**
 - `app-client`
3. Click **Finish** to complete the dialog. The POM editor opens. If there are references to other projects that were lost, you might see compilation errors.
4. Clean up compilation errors. Convert the projects that you want to reference to Maven, and then add Maven dependencies to those projects. See [Adding dependencies to other modules](#).
5. Update the project. After you change the project, right-click the project and select **Maven > Update Project**.

Converting EAR projects

About this task

Converting EAR projects involves several tasks:

1. Create a content folder.
2. Convert the EAR project.
3. Add modules to an EAR.
4. Add dependencies to other modules.
5. Add libraries to the EAR library directory.

Perform the following steps to complete these tasks.

Creating a content folder

About this task

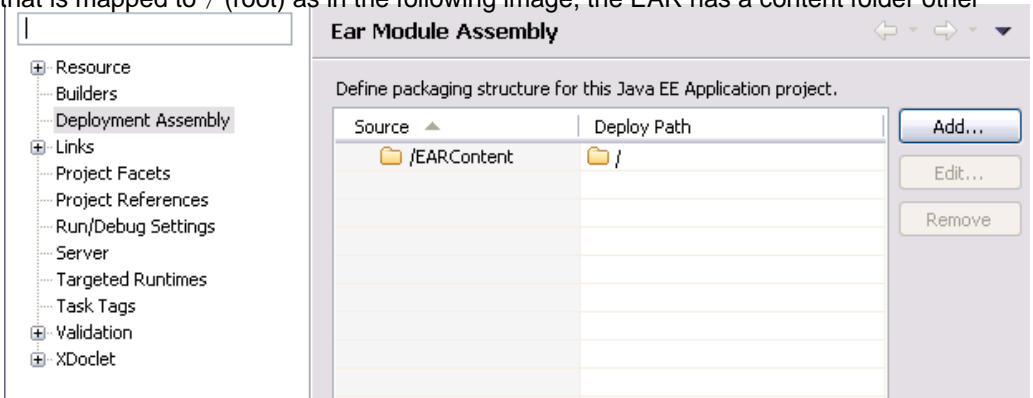
When an EAR project is created in WebSphere® Application Server Developer Tools using the default workspace preferences, the content directory of the EAR is empty, which means that everything in the root folder of the project is included in the EAR file. If the project does not have a content folder, you must create a content directory.

Procedure

1. Check whether the EAR has a content folder.

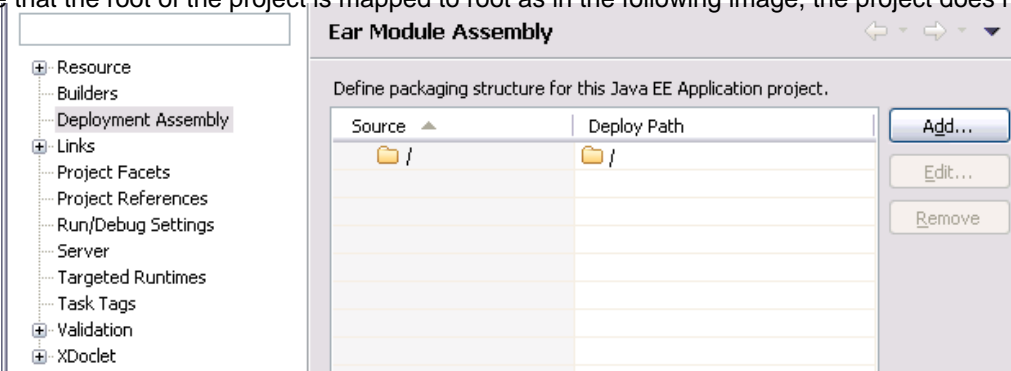
A. Right-click the project and select **Properties > Deployment Assembly**.

B. If you see a content folder that is mapped to / (root) as in the following image, the EAR has a content folder other



than the root of the project.

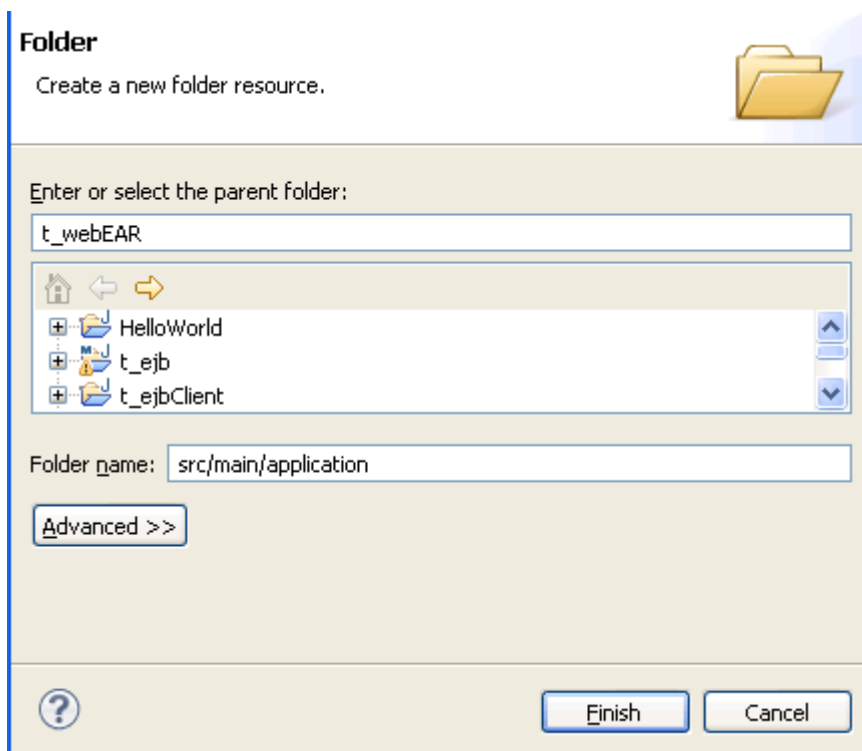
C. If you see that the root of the project is mapped to root as in the following image, the project does not have a content



directory:

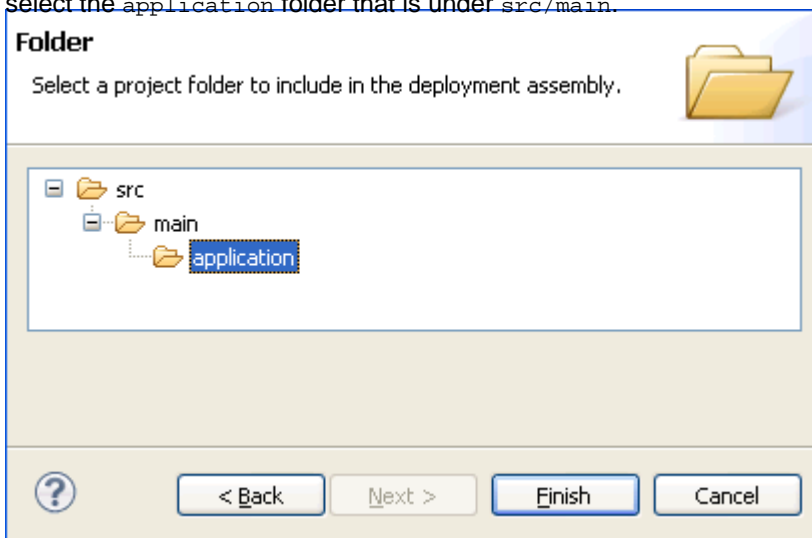
2. If the project does not have a content folder, click **Cancel** on the Deployment Assembly page and follow these steps to create one:

A. Right-click the EAR project and select **New > Folder**. In **Folder name**, enter the name of the folder structure. For Maven projects, the recommended folder structure is `src/main/application`. Click **Finish**.

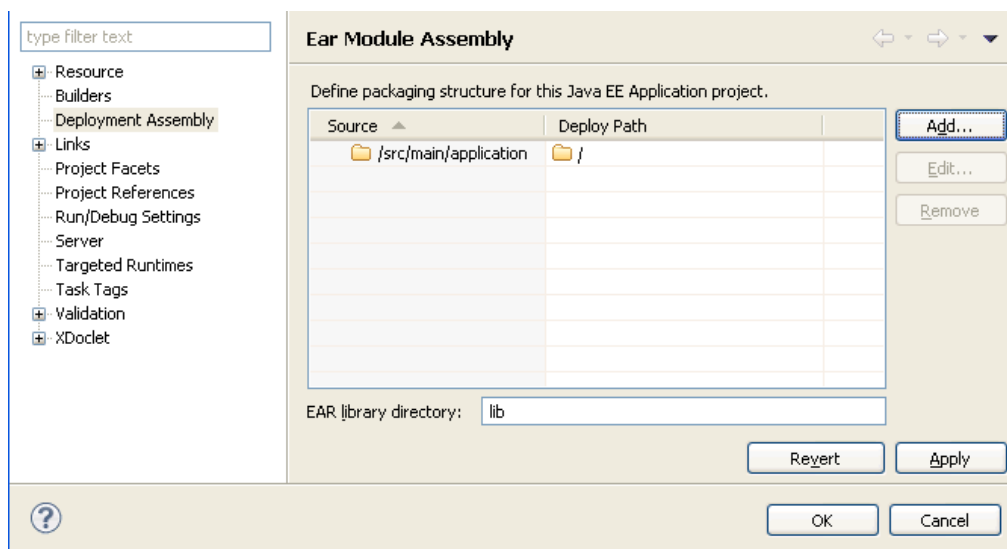


B. Access the deployment assembly page. Right-click the project and select **Properties > Deployment Assembly**.

- C. Select the entry that maps the root of the project to the root of the EAR and click **Remove**.
- D. Click **Add**, select **Folder**, and click **Next**.
- E. Select the folder that you created and click **Finish**. For example, if you created the recommended folder structure, select the application folder that is under `src/main`.



- F. Check the structure. In the deployment assembly section, you now have the folder that you created mapped to the root of the project. The following image shows the example if you created a `src/main/application` folder.



- G. Click **OK** to complete the deployment assembly mapping.
- H. Move all of the resources that you want to be included in the EAR from the root of the project to the folder that you created. For example, if you used the recommended folder structure `src/main/application`, and the EAR has a deployment descriptor, move the `META-INF` folder from the root of the EAR file to the `application` folder.

Converting the EAR project

About this task

When you convert a project, keep in mind the dependencies that you have to other artifacts and how the artifacts are accessed by the project. In non-Maven projects, this information is specified in project metadata and the `MANIFEST.MF` files directly. However, in Maven projects, dependencies are specified in the `pom.xml` file. Note the modules that are part of the EAR, the libraries that are shared to other modules using the library directory (for EARs version 5 or newer), or other dependencies.

First, use the deployment assembly page to gather the dependencies information:

1. Right-click the project.
2. Select **Properties > Deployment Assembly**.
3. Make note of the modules that are part of the EAR.
4. Make note of the EAR library directory value if the EAR is version 5 or later.
5. Make note of the JAR files and projects in the library directory. To identify these files, look for JAR files and projects that have a deploy path that begins with the value of the library directory.

Note: Before you convert the project, save a copy of the EAR's deployment descriptor file, `META-INF/application.xml`, if it exists. Saving a copy is necessary because during the conversion the existing deployment descriptor is replaced by an empty deployment descriptor.

Procedure

1. Right-click the project and select **Configure > Convert to Maven Project**.
2. Complete the Maven POM dialog. Enter a **Group Id**, **Artifact Id**, and **Version** or accept the defaults. Optionally, enter a **Name** and **Description**. For **Packaging**, enter `ear`.
3. Click **Finish**. The POM editor opens.**Note:** If the EAR had a deployment descriptor, then a new, empty deployment descriptor is generated. If the original deployment descriptor contained elements such as security roles, configure the elements in the `pom.xml` file. For more information, see: [Maven EAR plugin](#)
4. Update the project. After you change the project, right-click the project and select **Maven > Update Project**.
5. To complete the EAR conversion, you must complete the following three tasks:
 - A. [Adding modules to an EAR project](#)
 - B. [Adding dependencies to other modules](#)
 - C. [Adding libraries to the EAR library directory](#)

Setting POM entries for projects that target WebSphere Application Server

Before you begin

Important: Applicable editions: Liberty profile, full profile

Right-click on the `pom.xml` file and select **Open with** > **Maven POM Editor**. Go to the `pom.xml` tab.

About this task

In addition to the recommended workspace preferences for Maven, you can set several POM entries that generate project structures and class path entries that are best suited for WebSphere® Application Server deployments.

Procedure

1. Set the output folder location. By default, web projects designate output locations that are not nested within the WAR resource folder. This output location breaks the single-root rule for rapid deployment. Specify the source and test output locations as shown in the following example to designate output locations within the resource folder:<build>

```
<outputDirectory>${project.basedir}\src\main\webapp\WEB-INF\classes</outputDirectory>
<testOutputDirectory>${project.basedir}\src\main\webapp\WEB-INF\classes</testOutputDirectory>
```

2. Set the <finalName> element. By default, Maven uses the version that is designated in the POM as part of the archiveName for both EAR and WAR modules projects. Set <finalName> with a value that equals the project name for both WAR and EAR projects. For example:<build>

```
<finalName>${project.name}</finalName>
```

3. Set the module URI and bundleFileName in the EAR POM. Similar to the <finalName> for stand-alone EAR and WAR archives, the default names for EAR modules are derived from the version information. Designate the settings in the EAR POM file to be consistent with the project name. For example, if the project name was SimpleWeb, use the following settings:<modules>

```
<webModule>
  <groupId>test</groupId>
  <artifactId>SimpleWeb</artifactId>
  <uri>SimpleWeb.war</uri>
  <bundleFileName>SimpleWeb.war</bundleFileName>
```

Parent topic: [Developing Apache Maven projects](#)

Related tasks:

[Setting Maven preferences](#)

Defining Java EE module dependencies

For non-Maven Java™ EE projects, the Deployment Assembly property page is used to define dependencies to other projects, libraries, and utilities. The page is also used to describe the assembly of the archive at run time, including assembling modules within an EAR. However, Maven projects define dependencies within the `pom.xml` file and generate and derive dependencies that are used by the project and publishing mechanisms. All dependencies must be defined within the `pom.xml` file. Dependencies can be defined in source under the `<dependencies>` tag or on the POM editor dependencies tab.

About this task

Important: Applicable editions: Liberty profile, full profile

- [Adding dependencies to other modules](#)

You can add dependencies to other modules by setting the `addClasspath` element to `true` in the `pom.xml` file or by using the POM editor.

- [Adding modules to an EAR project](#)

You can add modules that are Maven projects or Maven archives to an EAR project.

- [Adding libraries to the EAR library directory](#)

Parent topic: [Developing Apache Maven projects](#)

Related tasks:

[Adding dependencies to other modules](#)

[Adding modules to an EAR project](#)

[Adding libraries to the EAR library directory](#)

Adding dependencies to other modules

You can add dependencies to other modules by setting the `addClasspath` element to `true` in the `pom.xml` file or by using the POM editor.

About this task

Important: Applicable editions: Liberty profile, full profile

Procedure

1. Set `addClasspath` to `true` in the `pom.xml` source.

- A. Open the `pom.xml` file and access the source view. To access the source view, right-click the `pom.xml` file in the project and select **Open With > Maven POM Editor**. Click the **pom.xml** tab.
- B. Find the correct `<plugin>` section to update. Look for the `<plugin><artifactID>` section that corresponds to your project type. For example, a web project that was converted to a Maven project has the `artifactID` set to `maven-war-plugin`. A converted EJB project will have the `artifactID` set to `maven-ejb-plugin`. The following example shows the `<plugin>` section with the `artifactID` `maven-war-plugin`:`<plugin>`

```
<artifactId>maven-war-plugin</artifactId>
<version>2.2</version>
<configuration>
  <warSourceDirectory>WebContent</warSourceDirectory>
  <failOnMissingWebXml>>false</failOnMissingWebXml>
</configuration>
</plugin>
```

- C. Update the `<configuration>` section to set `addClasspath` to `true`. Add the following code inside the configuration section:`<archive>`

```
<manifest>
  <addClasspath>>true</addClasspath>
</manifest>
</archive>
```

The following example shows a completed section for a `maven-war-plugin` artifact with `addClasspath` set to `true`.

```
<plugin>
<artifactId>maven-war-plugin</artifactId>
<version>2.2</version>
<configuration>
  <warSourceDirectory>WebContent</warSourceDirectory>
  <failOnMissingWebXml>>false</failOnMissingWebXml>
  <archive>
    <manifest>
      <addClasspath>>true</addClasspath>
    </manifest>
  </archive>
</configuration>
</plugin>
```

2. Use the Dependencies tab of the POM editor to add the dependency. Click the **Dependencies** tab. Click **Add** to access the select dependency dialog. Enter the values for the module that you want to add as a dependency. Click **OK** when finished. **Note:** When a dependency is added to a web project, the dependency is included by default in `WEB-INF/lib`

location. If the maven-war-plugin is configured to add the class path to the `MANIFEST.MF` file, then every dependency is included in both the `MANIFEST.MF` file and the `WEB-INF/lib` location. If you want the dependency to be included only in the `MANIFEST.MF` file, mark the dependency as optional. To mark a dependency as optional, in the **Dependencies** tab, select the dependency, click the **Properties** button. In the properties, check the **Optional** box and click **OK**.

Parent topic: [Defining Java EE module dependencies](#)

Adding modules to an EAR project

You can add modules that are Maven projects or Maven archives to an EAR project.

About this task

Important: Applicable editions: Liberty profile, full profile

Procedure

1. Access the Add dependency dialog. Open the POM editor by double-clicking the `pom.xml` file. Click the **Dependencies** tab, then click **Add**.
2. Enter values for the module. Enter the **Group Id** and **Artifact Id** of the module that you want to add. Alternatively, you can use the filter field to search for the module.
3. Select the module, and click **OK**.
4. Configure the module in the maven-ear-plugin. Click the **pom.xml** tab to work directly with the POM source. Configure the module. The following example shows a configuration for a web module. For a list of supported modules, see <http://maven.apache.org/plugins/maven-ear-plugin/modules.html>.

```
<plugin>  
<artifactId>maven-ear-plugin</artifactId>  
<version>2.7</version>  
<configuration>  
  <version>6</version>  
  <modules>  
    <webModule>  
      <groupId>sample</groupId>  
      <artifactId>webapp</artifactId>  
      <!-- More configuration can be set here -->  
    </webModule>  
  </modules>  
</configuration>  
</plugin>
```

5. Save the `pom.xml` file.
6. Update the project. Right-click the EAR project, and select **Maven > Update Project**. Click **OK** when you are finished.
7. Verify that the module was added to the deployment assembly. Right-click the project and select **Properties**. Click **Deployment Assembly**.
8. If the EAR has a deployment descriptor file, `META-INF/application.xml`, verify that the module was added to this file.

Parent topic: [Defining Java EE module dependencies](#)

Adding libraries to the EAR library directory

About this task

Important: Applicable editions: Liberty profile, full profile

Procedure

1. Open the `pom.xml` file in the POM editor by double-clicking the file. Click the **pom.xml** tab to work directly with the source.
2. Set the `lib` directory to be used in the EAR project. Add the element `<defaultLibBundleDir>` to the configuration of the `maven-ear-plugin` as in this example:`<plugins>`

```
<plugin>
  <artifactId>maven-ear-plugin</artifactId>
  <version>2.7</version>
  <configuration>
    <version>6</version>
    <defaultLibBundleDir>lib</defaultLibBundleDir>
  </configuration>
</plugin>
</plugins>
```

3. Save the `pom.xml` file.
4. Use quick fixes to resolve any errors. After you save the `pom.xml` file, you might see the following errors in the **Problems** or **Markers** view:

- Library Directory "`<PATH_TO_LIB_FOLDER>`" does not exist.
 - Project configuration is not up-to-date with `pom.xml`. Run Maven->Update Project or use Quick Fix.
- Right-click the errors and select **Quick Fix** to resolve the errors.

5. Place JAR files in the `lib` directory. If the `lib` directory is specified for the EAR project in the `<defaultLibBundleDir>` element, then all dependencies to JAR artifacts are bundled in the `lib` directory by default. However, if you want to place a JAR file in a different location, you can use the `<jarModule>` element to specify a different location. For example, the following fragment of `pom.xml` shows that the EAR project uses `lib` as the default directory to bundle JAR files:

```
<plugin>
  <artifactId>maven-ear-plugin</artifactId>
  <version>2.7</version>
  <configuration>
    <version>6</version>
    <defaultLibBundleDir>lib</defaultLibBundleDir>
```

In the following example, the `pom.xml` file has a dependency to a shared library that is packaged in `lib` and a dependency to an EJB client JAR file that is bundled at the root of the EAR file. Without further changes, both are bundled in the `lib` directory by default.`<dependencies>`

```
<dependency>
  <groupId>testapp</groupId>
  <artifactId>ejbclient</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
<dependency>
```

```
<groupId>testapp</groupId>
<artifactId>SharedLib</artifactId>
<version>0.0.1-SNAPSHOT</version>
</dependency>
</dependencies>
```

The following example shows how to change the location of the EJB client JAR file:<plugin>

```
<artifactId>maven-ear-plugin</artifactId>
<version>2.7</version>
<configuration>
  <version>6</version>
  <defaultLibBundleDir>lib</defaultLibBundleDir>
  <modules>
    <jarModule>
      <groupId>testapp</groupId>
      <artifactId>ejbclient</artifactId>
      <bundleDir>/</bundleDir>
    </jarModule>
  </modules>
</configuration>
</plugin>
```

This example shows that the default bundle directory for JAR files is `lib`, but the EJB client JAR file for `<artifactId>ejbclient</artifactId>` is in the root of the project. The bundle directory is indicated with `<bundleDir>/</bundleDir>`.

Parent topic: [Defining Java EE module dependencies](#)

Setting Maven preferences

For Maven projects, you can set preferences according to the requirements of the project.

About this task

Important: Applicable editions: Liberty profile, full profile

The preferences (**Window > Preferences**) to set for Maven can be set from a single preference page (**Java EE > Maven > Maven Project Settings**). However, these preferences are also located across the following preferences pages:

- **Java EE**
- **Java EE > Project**
- **Maven > Java EE Integration**

Procedure

To set or review preferences, follow these steps:

1. Click **Window > Preferences** to open the Preferences page.
2. Expand **Java EE > Maven** and select **Maven Project Settings**.
3. To set all the preferences in one action click **Set all Maven values**. To set individual preferences, click **Set Maven value** next to each preference. The following information lists the recommended values for Maven and other preference pages that are also updated when setting preferences in this page.
 - The following preferences are also found on the **Java EE** page:
 - **Use Ear Libraries classpath container**
 - false
 - **Use Web App Libraries classpath container**
 - false
 - The following preferences are also found on the **Java EE > Project** page:
 - **Enterprise Application membership > Add project an EAR**
 - false
 - **Enterprise Application project > Content Directory**
 - `src/main/application`
 - **Dynamic Web Project**
 - **Default Source Folder:** `src/main/java`
 - **Output Folder:** `src/main/webapp/WEB-INF/classes`
 - **Content Directory:** `src/main/webapp`
 - **EJB Project**
 - **Default Source Folder:** `src/main/java`
 - **Output Folder:** `target/classes`
 - **Application Client Project**
 - **Default Source Folder:** `src/main/java`
 - **Output Folder:** `target/classes`
 - **Connector Application Project**
 - **Default Source Folder:** `src/main/java`
 - **Output Folder:** `target/classes`
 - **Utility/JPA Projects**

- **Default Source Folder:** `src/main/java`

- **Output Folder:** `target/classes`

- The following preferences are also found on the **Maven > Java EE Integration** page:

- **Generate application.xml under the build directory**

- false

- **Maven Archiver generates files under the build directory**

- false

4. Click **Apply** to apply the changes, and click **OK** to close the Preferences page.

Parent topic: [Developing Apache Maven projects](#)

Accessing the Maven Console

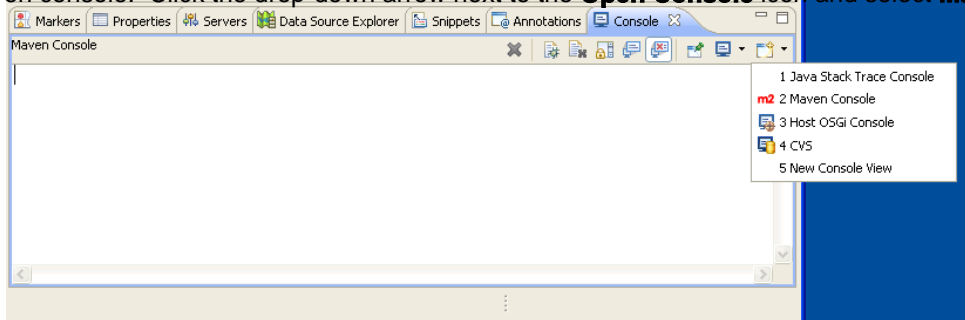
About this task

Important: Applicable editions: Liberty profile, full profile

The Maven console displays information for troubleshooting Maven projects. To access the console, follow these steps:

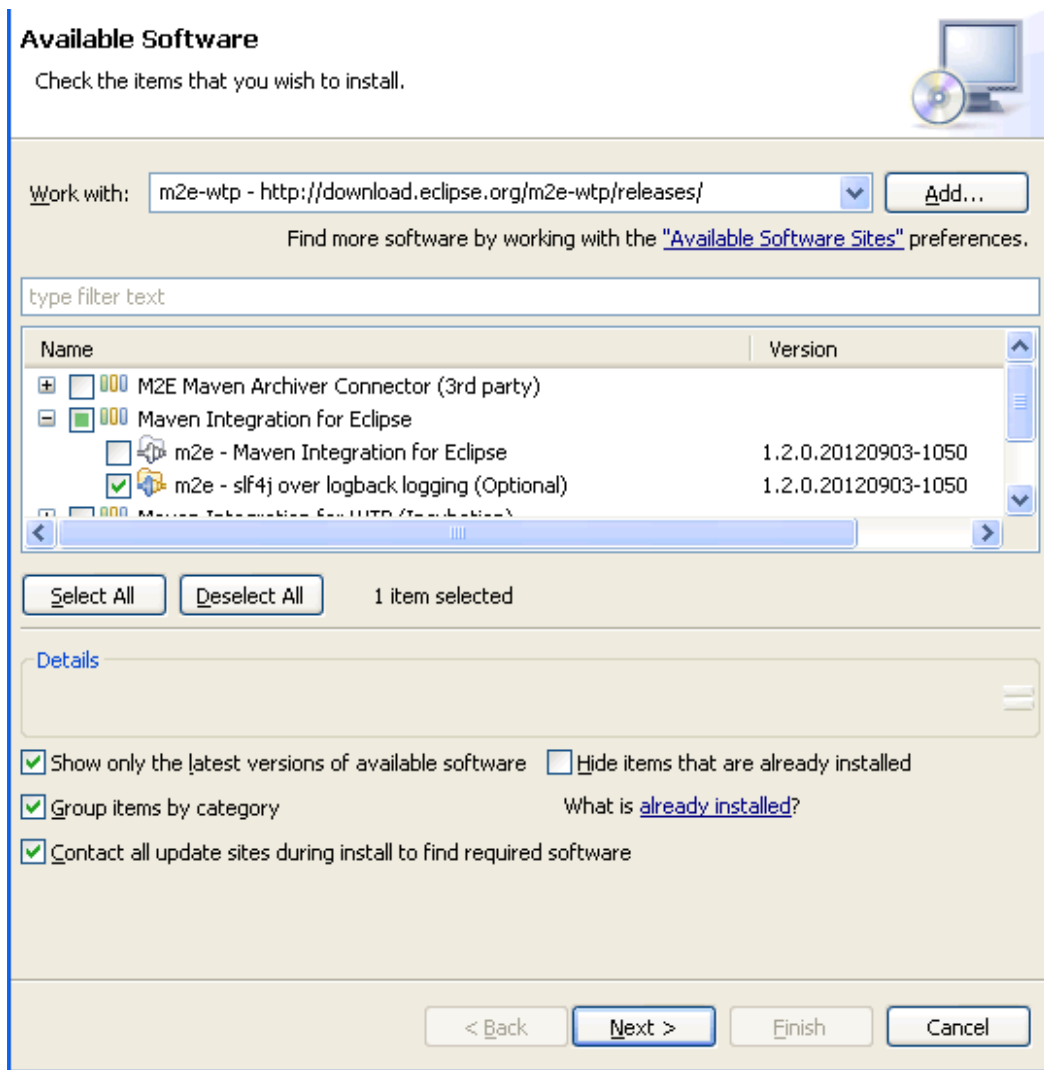
Procedure

1. Access the console view. Select **Window > Show View > Console**. The console view is displayed.
2. Select the Maven console. Click the drop-down arrow next to the **Open Console** icon and select **Maven Console**. The



console opens.

3. Enable logging. By default, the Maven Console does not show anything. To see content that is logged to the Maven Console, you must install the logging feature.
 - A. Select **Help > Install New Software**.
 - B. Click **Add** next to **Work with**, enter a name for the repository and enter `http://download.eclipse.org/m2e-wtp/releases/` in the location and click **OK**.
 - C. Ensure **Group items by category** checkbox is selected. Under **Maven Integration for Eclipse**, select **m2e - slf4j over logback logging (Optional)**. Click **Next** twice.



D. Review the license. If you want to accept the license, select **I accept the terms of the license agreement**. Click **Finish**.

E. After the software installs, restart the development workbench.

Parent topic: [Developing Apache Maven projects](#)

Example archetypes

Important: Applicable editions: Liberty profile, full profile

Archetypes are Maven templates that you can use to create projects that are based on set parameters. Using archetypes, you can quickly generate working projects for Java™ EE modules. The following tables list some archetypes that you can use. **Note:** These archetypes are not supported by IBM®. This information is provided for reference only. For more information about these archetypes, see <http://docs.codehaus.org/display/MAVENUSER/Home>

Java EE 6

Group ID	Artifact ID	Project Type
org.codehaus.mojo.archetypes	appclient-javaee6	Application Client
org.codehaus.mojo.archetypes	ear-javaee6	Enterprise Application
org.codehaus.mojo.archetypes	ejb-javaee6	EJB
org.codehaus.mojo.archetypes	webapp-javaee6	Web

Java EE 5

Group ID	Artifact ID	Project Type
org.codehaus.mojo.archetypes	appclient-jee5	Application Client
org.codehaus.mojo.archetypes	ear-jee5	Enterprise Application
org.codehaus.mojo.archetypes	ejb-jee5	EJB
org.codehaus.mojo.archetypes	webapp-jee5	Web

Java EE 1.4

Group ID	Artifact ID	Project Type
org.codehaus.mojo.archetypes	ear-j2ee14	Enterprise Application
org.codehaus.mojo.archetypes	ejb-j2ee14	EJB
org.codehaus.mojo.archetypes	webapp-j2ee14	Web

Java EE 1.3

Group ID	Artifact ID	Project Type
org.codehaus.mojo.archetypes	ejb-j2ee13	EJB
org.codehaus.mojo.archetypes	webapp-j2ee13	Web

Parent topic: [Developing Apache Maven projects](#)

Developing web applications

You can develop web applications, including static web pages and dynamic web applications. These topics contain instructions for creating, testing, and publishing your web applications.

- [Learn about web applications](#)

The workbench provides tools to develop web applications, such as static web pages, websites, and complex applications that use JavaServer Faces technologies to access data. Your web application contains all of the resources that are required to realize your business requirements. It also contains deployment descriptors to test and publish your web application to a server.

- [Web development tools](#)

- [Creating web projects](#)

Web projects hold all of the web resources that you create, maintain, and use as you develop your web application. The web project is the environment where you perform activities such as link-checking, building, testing, and publishing.

- [Creating website styles](#)

Websites and web pages are a part of every web application. Each web page helps to achieve the overall goal of the entire website through its content and design.

- [Creating and importing web pages and resources](#)

After you create a web project, you can populate your web application with the required web pages and web resources that complement the high-level goal and supporting goals of your web application.

- [Developing mobile web applications](#)

The mobile web application development tools help you to create interactive web pages that are optimized for mobile devices.

- [Adding tag library support for web projects](#)

This product supports HTML, JSP, and JSF tag libraries. You can also add custom or third-party tag libraries to your web projects.

- [Debugging web applications](#)

You can debug your web applications on a server to detect and diagnose errors in your application.

- [Configuring web applications](#)

You configure web applications through the deployment descriptor or through annotations.

- [Testing and publishing web applications](#)

The testing and publishing tools provide runtime environments where you can test JSP files, servlets, HTML files, enterprise beans, and Java™ classes.

- [Adding data using Java Persistence API \(JPA\)](#)

Java Persistence API (JPA) is a specification for the persistence of Java objects to relational databases. You can use JPA to manage relational data in enterprise applications.

- [Editing source code](#)

You can use structured text editors to create, modify, and validate your source code.

- [Web application tutorials, samples, and videos](#)

To view the samples and tutorials in this product, click **Help > Help Contents** and expand the Samples and Tutorials sections. To view the videos, click the links. Learn about different aspects of web application development from the following videos, samples, and tutorials:

Learn about web applications

The workbench provides tools to develop web applications, such as static web pages, websites, and complex applications that use JavaServer Faces technologies to access data. Your web application contains all of the resources that are required to realize your business requirements. It also contains deployment descriptors to test and publish your web application to a server.

- [Overview](#)
- [Getting Started](#)
- [Samples and Tutorials](#)
- [Web resources for learning](#)

Overview

You can read the following topics before creating a web application for planning and technology overview information. These topics are useful if you are new to web application development or developing web applications in this development environment.

-  [Web projects](#)
-  [JavaServer Pages \(JSP\) technology](#)
-  [The Dojo Toolkit](#)

Getting Started



If you are already familiar with web development technology, the following topics guide you through the development process.

-  [Creating web projects](#)
-  [Creating and importing web pages and resources](#)
-  [Creating Web 2.0 applications](#)
-  [Developing mobile web applications](#)
-  [Testing and publishing web applications](#)
-  [Tag library support for web projects](#)

Samples and Tutorials

For a complete list of web application development samples and tutorials, see [Web application tutorials, samples, and videos](#). To view the samples and tutorials in this product, click **Help > Help Contents** and expand the Samples and Tutorials sections.

Learn about different aspects of web application development from the following samples and tutorials:

-  **Sample: JPA JSF Employee List application**
 - This sample demonstrates the use of Java Persistent API (JPA) to access employee data. The application that is used in this example maps persistent entities to columns in a database table. Information is displayed on a website where users can create, read, update, and delete data.
-  **Tutorial: Create a Web 2.0 application with Dojo**
 - This tutorial provides you with the opportunity to explore the different tools provided by this product to support rapid development of Dojo web applications.

Web resources for learning

In addition to the information found in this information center, the following links provide additional learning material.

-  [Rational® Application Developer wiki](#)
-  [IBM® Redbooks®: Rational Application Developer V8 Programming Guide](#)
-  [IBM Redbooks: Rational Application Developer V7.5 Programming Guide](#)
-  [IBM Redbooks: Rational Application Developer V7 Programming Guide](#)

Parent topic: [Developing web applications](#)

Web development tools

The web development tools enable you to create web applications that range from simple static sites to dynamic web applications that use the latest web technologies.

The web perspective contains a selection of views and editors that are customized to be most useful to a web developer. These views include reusable drag components, link structures, access to page data, your project resources, code snippets, and more.

The editors range from standard source editors with [content assist](#) features to more full featured editors, such as the following:

- [Rich Page Editor](#) helps you to create and edit HTML files, add Dojo widgets to HTML pages, and create and edit web pages for mobile devices.
- [Mobile browser simulator](#) helps you test mobile web applications without having to install device vendor native SDK.

- **[The web perspective](#)**

- **[Enterprise Explorer view and web development](#)**

- **[Palette view](#)**

- **[Rich Page Editor](#)**

- **[Properties view associated with Rich Page Editor](#)**

The Properties view that is associated with Rich Page Editor displays specific information for the currently selected tag in a web page. You can use the Properties view to edit properties that are related to the appearance of tags in a web browser. For example, you can change CSS style information, default attribute values, Dojo properties, and jQuery properties, as required.

- **[Mobile Navigation view](#)**

- **[Mobile browser simulator](#)**

The mobile browser simulator is a web application that helps you test mobile web applications without having to install device vendor native SDK.

Parent topic: [Developing web applications](#)

The web perspective

The web perspective combines views and editors that assist you with web application development. This is the perspective in which you typically edit web project resources, such as HTML and JSP files, and deployment descriptors.

The following are some of the views that are helpful when developing web applications in the web perspective.

- Colors

- Enables you to apply colors from a palette (or custom colors) to selected objects in the editing area.

- Mobile Navigation

- Use the Mobile Navigation view to manage Dojo mobile view widgets.

- Palette

- Contains expandable drawers of drag objects. Enables you to drag objects, such as tables or form buttons, onto the Design or Source page.

- Problems

- Shows list of errors, warnings, and information about an operation. For example, if there are errors when validating the code in an HTML file, the errors appear in this list.

- Properties

- Provides tabbed pages that enable you to update properties for tags that are selected in files open in the active web editor. Changes to property value text fields are reflected in the edited file immediately when cursor focus is changed, or when you press the Enter key. In addition, changes to any of the controls in the Properties view are immediately reflected in the edited file.

- Quick Edit

- Enables you to edit small bits of code, including adding and editing actions that are assigned to tags.

- Servers

- Lists servers that are defined for the project and their status.

- Services

- Lists all of the services available to all of your projects including web services and RPC Adapter services. This view does not require you to open a web page in the editor in order to view the services specific to that particular page.

- Snippets

- Contains expandable drawers of drag code snippets. For instance, you can add JavaScript macros that become part of the user interface (UI), like a script to display the date and time when the page was last updated. If you are working in a JSP file, you can add common JSP code. You can drag code snippets into the Design and Source pages.

- Styles

- Provides guided editing for cascading style sheets and individual style definitions for HTML elements.

- Tasks

- Shows a list of "to do" items that you create and to which you assign priorities. To create a task, right-click in the list and select **Add Task** from the menu.

- Thumbnails


- Shows thumbnails of the images in the selected project, folder, or file. This view is especially valuable when used to add images from the artwork libraries (included in the product) to your page designs. You can drag from this view into the Enterprise Explorer view or the Design or Source page of Page Designer.

There are other views in this perspective that you might find useful. To add more views, select **Window > Show View** from the menu bar. You can close, resize, or move any of the views.

Parent topic: [Web development tools](#)

Enterprise Explorer view and web development

The Enterprise Explorer view is the default view in the web perspective. This view provides the following notable features:

- VCM (version control management) information can be toggled from the Preferences page (**Window > Preferences > General > Appearance > Label Decorations**)
-  You can drag files from Windows Explorer or the desktop into the Navigator view.
- View filtering is supported by selecting **Filters** from the Navigator view **Menu**. Resources can be filtered by name, project type, or content type. Files beginning with a period are filtered out by default.
- The status line shows the full path of the selected resource.
- Dragging a .java file from the Navigator view into a JSP file inserts a usebean tag. The same behavior is exhibited when a .class file is dragged into a JSP file.
- Errors and warnings are indicated with a red or yellow warning next to the resource with the problem. Parent containers up to the project level (for all projects types) also show the red error or yellow warning indicators. Errors and warnings can include Java™, HTML/JSP, and Links Builder problems.
- Items available from the **New** cascading menu in the project menu are context-sensitive. All menus have **Project** and **Other** options.

Organization of the Enterprise Explorer view

The Enterprise Explorer view shows a custom view of all projects with added Java EE extensions. The following are some of the notable top-level objects that are displayed beneath the project node (based on default folder names).

- Web Deployment Descriptor

- This file corresponds to the WebContent/WEB-INF/web.xml file, which is used to specify deployment information for modules that are created in the web development environment. Use one of the following methods to edit this file:
 - Use the [Web Deployment Descriptor Editor](#).
 - Right-click the file and select **Open With** from the menu to open the file with a different editor.


- Java Resources

- This node shows Java resources within the project. If the project contains a single Java source folder, the packages and classes (for example, servlets, beans) within that folder are displayed directly beneath the Java Resources folder node. If the project contains multiple source folders, each source folder is displayed beneath the Java Resources folder. You can expand source folders to show their packages and classes.

- Libraries

- This folder contains the library JAR files that are defined in the project properties. Three types of JAR files are shown:
 1. JAR files included in the WebContent/WEB-INF/lib directory of the project
 2. JAR files external to the project, such as `j2ee.jar` and `rt.jar`
 3. Project libraries, which are special references to a Java projectWhen a web project is exported, a JAR file is automatically created from the Java project to be used by the web application during run time. Libraries are shown in class path order. By default, only the JAR files that are contained within the project are shown. You can also show external JAR files and project libraries by selecting **Menu > Show Referenced Libraries** from the Navigator view.

- imported_classes folder

- This folder can be created during a WAR import, and contains class files that do not have accompanying source. The **imported_classes** folder is a Java classes folder; Java classes folders can also be created using the web project **Java Build Path** properties page.  You can drag class files from the Windows Explorer or desktop to the **imported_classes** folder in the Navigator view.

- **Web content folder**

- This folder contains items to be published to the server. By default, this folder is named **WebContent** for newly created web projects. **Note:** You can change the name in the creation wizard on the web facet page.

- **META-INF**

- This directory contains the `MANIFEST.MF` file. This file is used to map class paths for dependent JAR files that exist in other projects in the same Enterprise Application project. An entry in this file updates the runtime project class path and Java build settings to include the referenced JAR files.

- **Theme**

- The suggested directory for cascading style sheets and other style-related objects.

- **WEB-INF**

- The directory where supporting web resources for a web application are kept (for example: `.xmi` files, `.xml` files, and `web.xml`.) **Tip:** You can double-click the `web.xml` file to open the Web Deployment Descriptor editor. You can also open the Web Deployment Descriptor editor by double-clicking the Deployment Descriptor.

Parent topic: [Web development tools](#)

Parent topic: [Creating web projects](#)

Palette view

The Palette view (**Window > Show View > General > Palette**) contains a palette with a series of *drawers*. Each drawer contains items that you can drag into the active editor. For example, if you are editing a JSP page in Rich Page Editor, you can open the JavaServer Pages drawer and drag a JSP bean onto the JSP page. The drawers and their contents vary depending on the active editor.

Customizing the palette

To customize the Palette view, right-click within the Palette view, select **Customize** and then add or hide the items that you want to see. Additionally, you can increase the size of the icons in the palette by right-clicking within the palette and selecting **Use Large Icons**).

Parent topic: [Web development tools](#)

Rich Page Editor

Use Rich Page Editor to easily edit HTML files, add Dojo widgets to HTML pages, and create and edit web pages for mobile devices. Rich Page Editor is a multi-tabbed editor that provides multiple views to show different representations of your page.


Views

You can use the Source, Design, and Split views in Rich Page Editor to view and work with your files or pages. Each view in Rich Page Editor works with several other views and tools that are included in the web perspective, including the following interface elements:

- Mobile Navigation, Outline, and Properties views
- Toolbar buttons
- Menu bar options
- Pop-up (right-click) menus
- Palette components

Table 1. Rich Page Editor views

Editor view	Description
Source	The Source view helps you to view and work directly with the source code of a file. The Mobile Navigation, Palette, Outline, Page Data, and Properties views have features that supplement the Source view.
Split	The Split view combines the Source and Design views in a split screen view. Changes that you make in one part of the split screen are automatically updated in the other part. You can split the view horizontally or vertically.

A screenshot of the Rich Page Editor toolbar is shown at the bottom of the table. The toolbar contains various icons for editing and viewing. The icon representing the Split view, which shows two vertical panels, is circled in red.

Design

The Design view is a WYSIWYG environment. This view helps you to create and work with a file while viewing how your web page and dynamic content might look on a mobile device. You can use this view to visually edit files. For example, the Design view includes features that you can use to complete the following tasks: Drag items from the Palette and Enterprise Explorer views. Rotate the screen orientation when you use a mobile device profile to view your mobile web page in either portrait or landscape mode. Scale the mobile device to fit the size of the current Design view. Using this feature, you can see the entire visual canvas without the need to scroll. View how your page is displayed on different devices by selecting a device from the device list. The selected device specifies the size of the mobile device that you want to view and affects the size of the Design view area. View how your mobile web page is displayed in different styles. For example, Android, iPhone, or BlackBerry. By choosing a particular skin, you can switch to another device-specific style to view the layout and appearance of your page as it would appear on this specific device. **Note:** The **Skin** list is available only for MobileFirst application pages.

Design Mode editing

You can use the Design Mode editing features of Rich Page Editor to add and edit widgets in the Design view. To enable the Design Mode editing features, click the **Design Mode** icon.



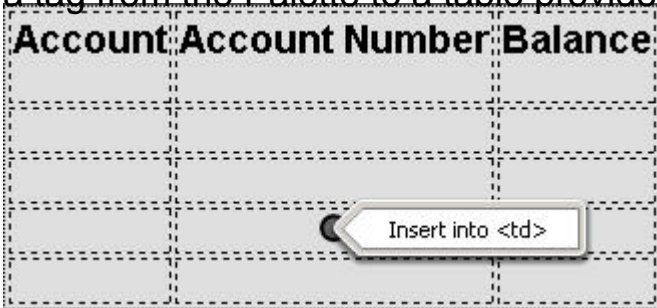
The following screen capture shows what a table looks like in the Design view of Rich Page Editor when Design Mode is enabled.

Account	Account Number	Balance

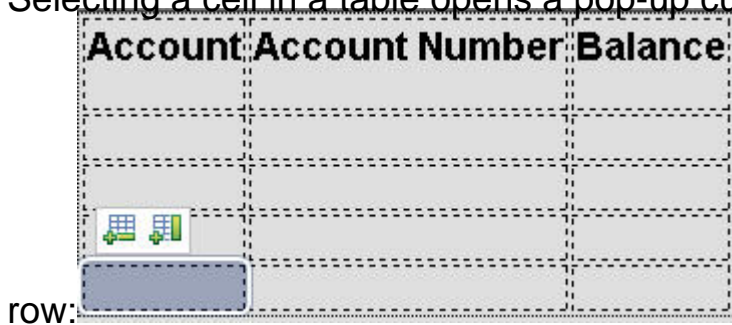
The following screen capture shows what the same table looks like in the Design view of Rich Page Editor when Design Mode is not enabled.

Account Account Number Balance

The Design Mode editing features guide the placement of code when you drop a widget on a container widget. Visual cues highlight the possible drop locations and pop-up cues indicate the editing function that is available for the selected widget. Design Mode also adds dashed borders to empty table cells. For example, dragging a tag from the Palette to a table provides a visual cue for placement:



Selecting a cell in a table opens a pop-up cue that you can use to add a column or



Properties view associated with Rich Page Editor

The Properties view that is associated with Rich Page Editor displays specific information for the currently selected tag in a web page. You can use the Properties view to edit properties that are related to the appearance of tags in a web browser. For example, you can change CSS style information, default attribute values, Dojo properties, and jQuery properties, as required.

You can use the Properties view to edit JavaScript, HTML, or JSP tags when the Design, Source, or Split view is open in Rich Page Editor. Changes in the Properties view are displayed in Rich Page Editor when you change the cursor focus or press Enter. If you update tags in the Source view of Rich Page Editor, your changes take effect immediately in the Properties view.

Breadcrumb navigation

When you select a node in Rich Page Editor, the Properties view uses a breadcrumb trail to provide context for the selected node:



html > body > div > div > ul > li > a

You can scroll through the breadcrumb trail without losing the position of your cursor in Rich Page Editor. Using this feature, you can quickly update the properties of ancestor elements.

Categorized property pages

The Properties view organizes properties into various categories, including:

- Styles

- Use to manipulate basic CSS style information (such as an attribute or the class that is associated with it) or various font, color, and alignment properties.

- Layout

- Use to configure properties that control the layout of the element within the presentation of the page.

- All

- Use to view all of the attributes for an element, in a tabbed list.

- Dojo

- Use to configure Dojo-specific properties on certain widgets.**Note:** This category applies only to Dojo-enabled web projects.

- jQuery

- Use to configure jQuery-specific properties on certain widgets.**Note:** This category applies only to jQuery-enabled projects.

Parent topic: [Web development tools](#)


Mobile Navigation view

You can use the Mobile Navigation view to manage Dojo mobile view widgets and jQuery mobile web page widgets.

For example, by using the Mobile Navigation view, you can:

- Add or remove mobile views and pages.
- Switch visibility from one mobile view or page to another.
- Rename mobile views and pages.
- Set the default mobile view or page that is shown the first time that a web page opens.
- Link mobile views or pages.

The Mobile Navigation view is available from both the Web perspective and Rich Page Editor:

- To open the view from the Web perspective, select **Window > Show view > Other > Web > Mobile Navigation**.
- To open the view from Rich Page Editor, on the toolbar click **Show/Hide Mobile Navigation**: 

A mobile web page can contain multiple views or pages. You can create these views and pages inline or in external files.

- Inline mobile view or page

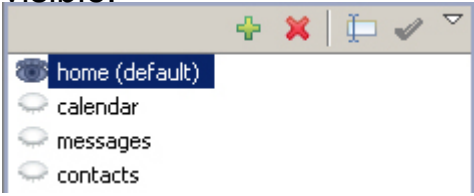
- A mobile view or page that is written within the source code of the mobile web page.

- External mobile view or page

- A mobile view or page that is written in a separate file or fragment. Creating mobile views or pages in separate files or fragments makes source code shorter and easier to manage.

When you open a mobile web page in Rich Page Editor, the mobile views or pages that are contained within that web page are displayed in the Mobile Navigation view. The icon to the left of each of the mobile views and pages indicates which one is visible in Rich Page Editor. If the mobile web page references external mobile views or pages, they are displayed in the Mobile Navigation view with a decorated icon. To open a new instance of Rich Page Editor for an external mobile view or page, double-click the mobile view or page.

What you can do in the Mobile Navigation view	Description
---	-------------

<p>Create mobile views or pages</p>	<p>You can create the following types of Dojo widgets:ViewA container that represents the entire mobile device screen.ScrollableViewA view widget with touch scrolling capability that you can use to provide fixed position header and footer bars.SwapViewA view widget that you can swipe horizontally to show adjacent SwapView widgets. You can create the following types of jQuery widgets:PageA container that represents the entire mobile device screen.Dialog pageA container that is shown in the form of a dialog box.</p>
<p>Switch between mobile views or pages</p>	<p>You can switch visibility between views or pages to specify which view or page is available in Rich Page Editor. In the following screen capture, the home view is visible in Rich Page Editor; the calendar, messages, and contacts views are not visible.</p>  <p>To switch to the calendar view, click the icon to the left of calendar.</p>
<p>Rename mobile views or pages</p>	<p>Right-click the view or page that you want to rename and then click Rename. For example, to rename the calendar view to internet:Right-click calendar and then click Rename.In the Mobile View id field, specify <i>internet</i>.</p>
<p>Set the default mobile view or page</p>	<p>Right-click the view or page that you want to set as the default and click Set as default.</p>
<p>Remove mobile views or pages</p>	<p>Right-click the view or page that you want to remove and click Remove.</p>

Link mobile views or pages	<p>You can link widgets, such as buttons or list view items, to mobile views or pages. You can drag a widget from the Design view in Rich Page Editor to a mobile view or page in the Mobile Navigation view. You can also drag mobile views or pages from the Mobile Navigation view to widgets in the Design view within Rich Page Editor. Tip: You can link Dojo mobile widgets to mobile views by using the Link to Mobile View action. In the Design view within Rich Page Editor, click the Dojo mobile widget that you want to link to a mobile view to open the toolbar. To open the Link to Mobile View dialog, on the toolbar click Link to Mobile</p> <p>View:  Select one of the following options. Click Inline Mobile View; from the list, select the mobile view that you want to link to the widget. Click Page Fragment, and then click Browse to browse to the mobile page file that you want to link to the mobile view. Click Finish.</p>
----------------------------	---

Mobile browser simulator

The mobile browser simulator is a web application that helps you test mobile web applications without having to install device vendor native SDK.

Important: The Mobile Browser Simulator supports the following web browsers:

- Firefox version 3.6 and later.
- Chrome 17 and later.
- Safari 5 and later

8.5.5.5 You can use the mobile browser simulator to preview applications on Android, iPhone, iPad, BlackBerry 6 and 7, BlackBerry 10, Windows Phone 8, and mobile web application environments.

8.5.5.6 BlackBerry 6 and BlackBerry 7 environments are deprecated.

8.5.5.5 Tip: When you preview an application on an Android, iPhone, iPad, BlackBerry 6 and 7, BlackBerry 10, or Windows Phone 8 environment, only the devices for the created environments are available. For example, if you preview a MobileFirst application on an Android environment, you can select from the list of available Android devices and also the device lists from any other environments added to your application.

You can also use the Ripple emulator to simulate the WebWorks API in your BlackBerry application. Using Chrome as your web browser, click **Open Simple Preview** in the simulator. A new tab opens in Chrome with your application loaded; you can open the Ripple emulator from this tab.

All environments can be previewed from the application folder. Each environment-specific preview allows for the addition of devices from available environments.






Skins can be tested per device in the mobile browser simulator. Only skins available for that platform is shown. A file can be saved in Worklight® Rich Page Editor and then instantly previewed by clicking **Go/Refresh**.

The link icon on the device toolbar can be selected to debug an application in a separate, simple preview.

Whenever a new environment or skin is added to an application, the mobile browser simulator must be restarted from Eclipse, **Run As > Run on Mobile Browser Simulator**.

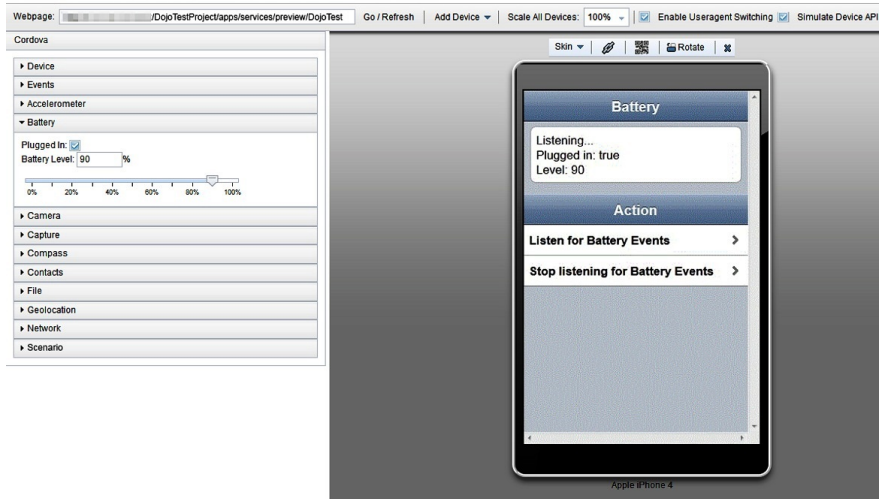
The Quick Response (QR) code icon on the device toolbar can be selected to show a QR code specific to the environment's URL. This QR code generator therefore allows for quick testing on a physical device.

8.5.5.5 The mobile browser simulator contains a frame that emulates a target device. It shows you what your page looks like inside the mobile device browser. You can switch the frame to emulate different screen resolutions and form factors, including BlackBerry 6 and 7, BlackBerry 10, Android, iPad, iPhone, and Windows Phone 8 mobile devices. You can also rotate the frame to mimic orientation change (portrait or landscape). You can add multiple devices to the frame to view the various displays simultaneously. If a device detection servlet is configured for your web project, the simulator emulates requests from different device-specific agents.

-  **Testing mobile applications**
-  **Calibrating the mobile browser simulator**
-  **Enabling user agent switching**
-  **Switching devices**
-  **Adding devices**

Mobile Browser Simulator

The Mobile Browser Simulator displays mobile web pages in a variety of mobile browser sizes and shapes.



Parent topic: [Web development tools](#)

Parent topic: [Testing mobile applications](#)

Creating web projects

Web projects hold all of the web resources that you create, maintain, and use as you develop your web application. The web project is the environment where you perform activities such as link-checking, building, testing, and publishing.

About this task

Use the Web Project wizard to create web projects. The following project templates are available in the Web Project wizard:

- Dojo Toolkit
- REST Services
- Simple
- JQuery

You can use the Simple project template to create any type of web project by adding features. **Tip:** To keep the configuration requirements to a minimum, use the project template that is most similar to the type of web project you want to create.

The following programming models are available for use when creating web projects:

- Java™ EE

- Select this option if you intend to use Java EE technologies such as servlets, JSPs, or JPA to develop your web application.

- Client-side only (HTML, JavaScript)

- Select this option if you intend to use only JavaScript, HTML, Dojo, and CSS to develop your web application. Server-side programming is not available with this option.

- OSGi

- Select this option if you intend to use Java EE technologies, such as servlets and JPA, within the OSGi programming model.

The following table lists the different types of web projects that you might want to create. The project template and available programming models to use when creating these web projects are also provided. *Table 1. Web projects*

Type of web project	Project template to use	Available programming models
Projects for mobile devices	Dojo Toolkit JQuery	Java EE Client-side only (HTML, JavaScript)
JPA-enabled projects	Dojo Toolkit REST Services Simple JQuery	Java EE OSGi
Web 2.0-enabled projects	Simple Dojo Toolkit	Java EE Client-side only (HTML, JavaScript) Note: The Client-side only programming model is not available for some of the Web 2.0 features. For example, it is not available for Ajax Proxy and Server-side Tech. OSGi
Dojo-enabled projects	Dojo Toolkit	Java EE Client-side only (HTML, JavaScript)

Procedure

1. Open the J2EE or Web perspective.
2. In the Enterprise Explorer view, right-click and select **New > Web Project**.
3. In the **Name** field, enter a name for your new web project.
4. From the list of project templates, select the project template to use when creating your web project.
5. In the Programming Model section, specify which of the available programming models to use when creating your web project. **Tip:** The Programming Model section is displayed only if there is more than one programming model available for use with the selected project template.
6. Click **Next** and complete the information that is required by the wizard to configure your new web project.

- **Web projects**

Use web projects to create and maintain the resources for your web applications. You can create web projects to manage content-based web applications that contain resources such as images and HTML files. You can also create web projects that contain dynamic files, such as JavaServer Pages or servlets.

- **Creating web projects for mobile devices**

Web projects hold all of the resources that are created and used when developing a mobile web application.

- **Creating JPA-enabled web projects**

JPA-enabled web projects hold all of the web resources that are used by your web application to access or modify data in a database.

- **Creating Web 2.0-enabled web projects**

Web 2.0-enabled web projects hold all of the web resources that are created and used when developing a Web 2.0 web application.

- **Web project features**

A project feature is a specific unit of function that you can add to a project when that function is required. When you add a project feature to a project, it can add natures, builders, class path entries, and resources, depending on the characteristics of the project. Features define characteristics and requirements for Java EE projects and are used as part of the runtime configuration.

- **Refactoring Web project content roots**

The context root is the web application root, which is the top-level directory of your application when it is deployed to the web server.

- **Enterprise Explorer view and web development**

Parent topic: [Developing web applications](#)

Web projects

Use web projects to create and maintain the resources for your web applications. You can create web projects to manage content-based web applications that contain resources such as images and HTML files. You can also create web projects that contain dynamic files, such as JavaServer Pages or servlets.

The structure of a web project mirrors the web application that is created from the project. The main project folder contains all of the development objects that are related to the web application. **Note:** In the Enterprise Explorer view, web projects are filtered into folder nodes to customize the display of web resources for easy management during development. For more information about this filtered structure, see [Enterprise Explorer view and web development](#).

The following table lists and describes the default elements that are in the web project folder hierarchy. *Table 1. Contents of the main web project folder*

Development object	Description
Web Deployment Descriptor	The standard web application deployment descriptor file (<code>web.xml</code>). This file describes how to deploy a module by specifying configuration and container options. A deployment descriptor file is automatically generated when you create a web project. This configuration file is used to run a servlet on an application server. If your web application does not contain any servlets, filters, or listeners, you can clear the Generate web.xml deployment descriptor check box in the Web Module configuration page when creating your web project. If you need to generate a deployment descriptor file later, right-click your web project and select Java EE > Generate Deployment Descriptor Stub .
JavaSource	This folder contains the Java™ source code for classes, beans, and servlets. When these resources are added to a web project, they are automatically compiled and the generated files are added to the WEB-INF classes directory. The contents of the source directory are not packaged in web application archive (WAR) files unless an option is specified when a WAR file is created.
imported_classes	Contains class files that do not have accompanying source. This Java classes folder is created while importing a WAR file. You can also use the Java Build Path properties page to create Java classes folders.

WebContent	Contains all of the web resources. For example, the HTML files, JSP files, and image files that are used to create a web application. If files are not placed in this directory, or in a subdirectory, the files are not available when the web application runs on a server. The folder structure represents the contents of the WAR file to be deployed to the server. Any files that are not in the WebContent folder are considered to be development-time resources, such as .java files, .sql files, and .mif files. These files are not deployed when the project is unit tested or published.
META-INF	Contains the MANIFEST.MF file that is used to map class paths for dependent JAR files that exist in other projects in the same Enterprise Application project. An entry in the MANIFEST.MF file updates the runtime project class path and Java build settings to include the referenced JAR files.
Themes	Contains cascading style sheets and other style-related objects.
WEB-INF	Contains the supporting web resources for a web application, including the <code>web.xml</code> file and the <code>classes</code> and <code>lib</code> directories. The structure of this directory is based on the <i>Sun Microsystems Java Servlet 2.5 and 3.0 Specifications</i> .
/classes	Contains servlets, utility classes, and the Java compiler output directory. The application class loader uses the classes in this directory to load the classes. Folders in this directory map package and class names. For example: <code>/WEB-INF/classes/com/corp/servlets/MyServlet.class</code> . The <code>.class</code> files are automatically placed in this directory when the Java compiler compiles Java source files in the Java Resources directory. Do not place any <code>.class</code> files directly into the <code>/classes</code> directory since they are deleted by the Java compiler when it runs.
/lib	Contains the supporting JAR files that are referenced by your web application. Your web application can use any classes in <code>.jar</code> files that are stored in this directory.

Libraries	<p>This folder mirrors the content of the /lib folder. It contains the supporting JAR files that are referenced by your web application and Web Library Projects. Web Library Projects are virtual JAR files that are not physically in the web project. Instead, these projects are associated with Java projects in a different location within your workspace. Web Library Projects are packaged with your project when you export the WAR file from your web application. Note: A library entry on the Java build path remains there unless the actual JAR file is deleted from the <code>WEB-INF/lib</code> folder. If you remove a library path entry but not the JAR file, the library entry is automatically added to the path again.</p>
-----------	--

Parent topic: [Creating web projects](#)

Creating web projects for mobile devices

Web projects hold all of the resources that are created and used when developing a mobile web application.

Procedure

1. In the Enterprise Explorer view, right-click and select **New > Web Project**.
2. In the **Name** field, enter a name for your new web project.
3. From the list of project templates, click **Dojo Toolkit**.
4. In the Programming Model section, specify one of the following programming models to use when creating your project:
 - To develop your web application using only Javascript, HTML, Dojo, and CSS, click **Client-side only (HTML, JavaScript)**. Server-side programming is not available with this option.
 - To use Java™ EE technologies such as Servlets, JSPs, or JPA to develop your web application, click **Java EE**.
5. To configure your new web project, click **Next**. The available configuration options depend on the programming model you selected.
6. If you are using the Java EE programming model, proceed to [step 7](#). If you are not using the Java EE programming model, jump to [step 9](#).
7. From the list of available configuration options, click **Deployment**.
8. From the **Target Runtime** list, select the run time to use at build time to compile your new web project. **Note:** If the target run time was previously set, this list is not available. Complete one of the following steps to make this list available:
 - Specify a new EAR project name and then select a target run time.
 - Select an existing EAR project from the list that uses the target run time that you want to use.
9. From the list of available configuration options, click **Dojo Toolkit**.
10. Specify that you want to install only the Dojo mobile components of the Dojo Toolkit:
 - A. Click **Change these setup options** and then click **Next** to proceed to the second page of the wizard.
 - B. Expand the **Select the Dojo components to be included in the project** section and ensure **dojo** is selected. Clear other components.
 - C. Expand **dojox** and select **mobile**.
 - D. Click **Finish** to install the Dojo mobile components.
11. Click **Finish** to create your web project.

Parent topic: [Creating web projects](#)

Creating JPA-enabled web projects

JPA-enabled web projects hold all of the web resources that are used by your web application to access or modify data in a database.

Procedure

1. In the Enterprise Explorer view, right-click and select **New > Web Project**.
2. In the **Name** field, enter a name for your new web project.
3. From the list of project templates, click one of the following templates to use as a starting point to create your JPA-enabled web project:
 - **Dojo Toolkit**
 - **REST Services**
 - **Simple**
 - **JQuery**
4. In the Programming Model section, specify one of the following programming models to use when creating your project:
 - To use Java™ EE technologies such as Servlets, JSPs, or JPA to develop your web application, click **Java EE**.
 - To use Java EE technologies such as Servlets and JPA within the OSGi programming model, click **OSGi**.
5. Click **Next** and from the list of available configuration options, click **Deployment**.
6. From the **Target Runtime** list, select the run time to use at build time to compile your new web project. **Note:** If the target run time was previously set, this list is not available. Complete one of the following steps to make this list available:
 - Specify a new EAR project name and then select a target run time.
 - Select an existing EAR project from the list that uses the target run time that you want to use.
7. Add the JPA (Java Persistence API) feature to your web project:
 - A. Click **Change Features**.
 - B. Select **JPA** and click **OK**. The JPA configuration is added to the list of available configuration options.
8. From the list of available configuration options, click **JPA**.
9. Create a connection profile that contains the connection property information that is required to connect to a data source in your enterprise:
 - A. Click **Add connection**.
 - B. From the list of Connection Profile Types, select your connection profile type and click **Next**.
 - C. Complete the information that is required by the wizard to create a connection profile.
 - D. Click **Finish** to return to the New Web Project wizard.
10. Click **Finish** to create your web project.

Parent topic: [Creating web projects](#)

Creating Web 2.0-enabled web projects

Web 2.0-enabled web projects hold all of the web resources that are created and used when developing a Web 2.0 web application.

Procedure

1. In the Enterprise Explorer view, right-click and select **New > Web Project**.
2. In the **Name** field, enter a name for your new web project.
3. From the list of project templates, click one of the following templates to use as a starting point to create your Web 2.0-enabled web project:
 - **Dojo Toolkit**
 - **Simple**
4. In the Programming Model section, specify one of the following programming models to use when creating your project:
 - To develop your web application using only Javascript, HTML, Dojo, and CSS, click **Client-side only (HTML, JavaScript)**. Server-side programming is not available with this option.
 - To use Java™ EE technologies such as Servlets, JSPs, or JPA to develop your web application, click **Java EE**.
 - To use Java EE technologies such as Servlets and JPA within the OSGi programming model, click **OSGi**.
5. To configure your new web project, click **Next**. The available configuration options depend on the programming model you selected.
6. If you are using the Java EE programming model, proceed to [step 7](#). If you are not using the Java EE programming model, jump to [step 9](#).
7. From the list of available configuration options, click **Deployment**.
8. From the **Target Runtime** list, select the run time to use at build time to compile your new web project. **Note:** If the target run time was previously set, this list is not available. Complete one of the following steps to make this list available:
 - Specify a new EAR project name and then select a target run time.
 - Select an existing EAR project from the list that uses the target run time that you want to use.
9. Add the Web 2.0 features to your web project:
 - A. Click **Change Features**.
 - B. Select **Web 2.0** and click **OK**. The Web 2.0 features define the characteristics of your Web 2.0-enabled web application. These features also specify the requirements and constraints that apply to your Web 2.0 project.
10. Click **Finish** to create your web project.

- [Installing the WebSphere Application Server Feature Pack for Web 2.0](#)

The WebSphere® Application Server Feature Pack for Web 2.0 provides a supported open Asynchronous JavaScript and XML (AJAX) development framework that uses existing SOA and Java EE assets to deliver rich internet applications.

- [Creating Dojo-enabled web projects](#)

Dojo-enabled web projects hold all of the web resources that are created and used when developing a Dojo web application.

- [Web 2.0 features](#)

The Web 2.0 features define the characteristics of your Web 2.0-enabled web application. These features also specify the requirements and constraints that apply to your Web 2.0 project.

Parent topic: [Creating web projects](#)

Installing the WebSphere Application Server Feature Pack for Web 2.0

The WebSphere® Application Server Feature Pack for Web 2.0 provides a supported open Asynchronous JavaScript and XML (AJAX) development framework that uses existing SOA and Java™ EE assets to deliver rich internet applications.

Before you begin

Important: Applicable edition: Full profile

Procedure

1. Open the IBM® Installation Manager.
2. Click **Install**. The Install Packages page opens.
3. In the package list, select **IBM WebSphere Application Server**. **Important:** If you are installing WebSphere Application Server Version 8.0 you also must install the IBM WebSphere Application Server Feature Pack for Web 2.0 and Mobile. Ensure that you select **IBM WebSphere Application Server Feature Pack for Web 2.0 and Mobile**.
4. Click **Next**.
5. Read the license agreements. Accept the license agreements then click **Next**.
6. Follow the instructions in the Installation Manager to install WebSphere Application Server. In the Features section of the Installation Manager, ensure that you select the WebSphere Application Server Feature Packs, if you are installing Feature Pack for web 2.0, if you are installing WebSphere Application Server Version 7.0, and IBM WebSphere Application Server Feature Pack for Web 2.0 and Mobile, if you are installing WebSphere Application Server Version 8.0.

Results

For more information about the Feature Pack for web 2.0, refer to [WebSphere Application Server Feature Pack for Web 2.0 and Mobile](#).

Parent topic: [Creating Web 2.0-enabled web projects](#)

About the IBM WebSphere Application Server Feature Pack for Web 2.0

About this task

The IBM WebSphere Application Server Feature Pack for Web 2.0 provides technology that can be used to create AJAX-style architectures. The feature pack is available with WebSphere Application Server 8.0, 7.0, and WebSphere Community Edition 2.0. The feature pack provides developers and architects, the resources to create AJAX web applications and architectures. The feature pack includes both client-side runtime and server-side functionality.

- Client runtime

- The Client-Runtime included with the feature pack consists of the technologies that are running on the browser-client. They include the open source Dojo Toolkit and a set of IBM extensions to the Dojo Toolkit to support additional functionality.

The [Dojo Toolkit](#) is a powerful open source JavaScript library that can be used to create rich and varied user interfaces running within a browser. The library requires no browser-side runtime plug-in and runs natively on all major browsers. This is boon for JavaScript developers since it helps abstract away the eccentricity of different browser implementations.

The open source Dojo Toolkit provided with the IBM's feature pack is divided into five sections:

- Base

- The Base is the kernel of the Dojo Toolkit and consists of dojo.js. The file is compact and optimized so as not take long to download to the browser. It contains the bootstrapping, useful utilities, event notification, to name just a few items.

- Core

- The Core, contains wide variety of graphical user interface widgets and the IO Transport for XHR requests to the server.

- Dijit

- Dijit builds on the Base and Core by providing a rich set of additional widget controls. The controls are internationalized and accessibility enabled.

- Dojox

- Dojox contains experimental aspects of the Dojo Toolkit and represents innovative material that might some day move into the base or Dijit modules. Dojox is an incubator of sorts and a preview of new features. Some of the modules in Dojox include charting, offline storage, and grid to name a few.

- Util

- The Util contains a testing harness for Dojo and can be used to test the widgets that are provided with the Dojo Toolkit.

- IBM Extension to the Dojo Toolkit

- In addition to the open source Dojo Toolkit for creating rich client side applications, IBM also provides a set of JavaScript extensions that developers find useful:

- **Atom Feed widget** - A client-side widget that can be used to render and use Atom syndication feeds.

- **IBM Gauge widgets** - A client-side widget that can be used to display numerical data in a graphically rich way.

- **IBM SOAP** - This extension can be used to connect a client-side browser widget to an existing SOAP-based service.

- **IBM Open Search library** - This extension enables you to invoke any Open Search-compliant service and to bind search results to widgets within your AJAX application.

- Server-side libraries and connectivity

- The feature pack also includes a rich set of libraries and connectivity features provided on the server to assist in client development. The features include:

- Ajax proxy

- The feature pack provides a Servlet-based forward proxy that can be used in the aggregation of content from different sites. To provide control, the proxy contains a white-listing configuration file that can be used to define the sites that the proxy can access. Additionally, the proxy can filter on HTTP headers, cookies, and mime-types to provide a level of control over the sites that a browser-based client can access.

- Web-remoting for Java Components

- A challenge in combining Ajax style architectures and Java EE is mapping client-side runtime to Java EE constructs. The feature pack provides a Remote Procedure Call Adapter (RPCAdapter) that is provided as a JAR library which can be embedded into a server-side web application. The RPCAdapter can be used to accept HTTP requests such as POST and GET and map the requests directly to user created classes. One of the powerful aspects of RPCAdapter is the ability to serialize EJB session and Collection data to a JSON or XML stream returned to the browser client. The JSON and XML data can contain the information to be displayed by the widget.

- Apache Abdera libraries

- Apache Abdera is an open source project providing feed syndication support. Abdera addresses both the Atom syndication format and the Atom publishing protocol. The Abdera libraries can used on the server to read syndication feeds from other sources or to generate your own feed content for use by your widgets.

- JSON4J

- The JSON4J library is an implementation of a set of JSON handling classes for use within Java environments. The library can be used to derive your own JSON data streams. The JSON4J library provides the following functions:

- A simple Java model for constructing and manipulating data to be rendered as JSON.
- A fast transform for XML-to-JSON conversion. JSON4J can be used to convert an XML reply from a web service into a JSON structure for easy use in an Ajax application.
- A JSON string and stream parser that can generate the corresponding JSONObject, which represents that JSON structure in Java.

- Web messaging service

- The web messaging service uses a publish and subscribe pattern to connect the browser to the WebSphere Application Server Service Integration Bus for server-side event push to the browser. Client/server communication is achieved through the Bayeux protocol. You can consider the web messaging service implementation as a comet server implementation. The Dojo Toolkit provides client-side support. Currently, the Dojo Toolkit is the only JavaScript library to support the Bayeux protocol, although any JavaScript library that implements Bayeux protocol support can communicate with web messaging service. The web messaging service server bridges browser clients to the Service Integration Bus, enabling a web service or any other item that is connected to the bus to publish events to web-based clients. You can use the web messaging service in a new or existing application by placing a utility file library JAR in an application web module, setting up a simple configuration file, and configuring Servlet mappings. The web messaging service is included in the Quote Streamer for WebSphere Application Server product samples.

Creating Dojo-enabled web projects

Dojo-enabled web projects hold all of the web resources that are created and used when developing a Dojo web application.

Before you begin

- [Install the WebSphere® Application Server Feature Pack for Web 2.0](#) to run your project on WebSphere Application Server.

Procedure

1. In the Enterprise Explorer view, right-click and select **New > Web Project**.
2. In the **Name** field, enter a name for your new web project.
3. From the list of project templates, click **Dojo Toolkit**.
4. In the Programming Model section, specify one of the following programming models to use when creating your project:
 - To develop your web application using only Javascript, HTML, Dojo, and CSS, click **Client-side only (HTML, JavaScript)**. Server-side programming is not available with this option.
 - To use Java™ EE technologies such as Servlets, JSPs, or JPA to develop your web application, click **Java EE**.
5. To configure your new web project, click **Next**. The available configuration options depend on the programming model you selected.
6. If you are using the Java EE programming model, proceed to [step 7](#). If you are not using the Java EE programming model, jump to [step 9](#).
7. From the list of available configuration options, click **Deployment**.
8. From the **Target Runtime** list, select a target run time for your new web project. **Note:** If the target run time was previously set, this list is not available. Complete one of the following steps to make this list available:
 - Specify a new EAR project name and then select a target run time.
 - Select an existing EAR project from the list that uses the target run time that you want to use.

By using the [Web Preview Server](#) you can compile, test, and run resources in a Dojo-enabled web project.

9. From the list of available configuration options, click **Dojo Toolkit**.
10. Configure how your project accesses the Dojo Toolkit and which version of the toolkit to use:
 - A. Click **Change these setup options**.
 - B. Choose one of the following options to set up Dojo in your web project and click **Next**:
 - **Copy Dojo into this project. It will be deployed from there.**
 - Specify the name of the Dojo folder and which of the following Dojo Toolkits to use:
 - One that is provided with the product
 - One that is in your workspace or file systemYou can select an archive file of a compressed Dojo distribution. When you click **Finish**, the contents of the archive file are automatically extracted into your project.
 - **Dojo is in a project in the workspace, and will be deployed from there.**
 - Browse to the root Dojo folder in another project in your workspace. The Dojo Toolkit is not copied into your project. It is deployed from the project that contains the toolkit.
 - **Dojo is remotely deployed or is on a public CDN.**
 - Specify the remote location of the Dojo toolkit source:
 - **Public CDN:** Enter the URL of a publicly available content delivery network. Content delivery networks provide geographically distributed hosting for open source JavaScript libraries. When a browser resolves the URL in your web application, the browser automatically downloads the file from the closest available server.

- **Remote URI:** Type the URI of the remote location to the root Dojo folder.

In the Corresponding Dojo section, choose the Dojo source distribution that is the closest match to your remote Dojo Toolkit. This selection provides access to tools such as content assist. You can choose the default Dojo provided with this product or browse to a Dojo folder in your workspace or file system.

C. If you copied the Dojo Toolkit into your project, you can include only the parts of the Dojo Toolkit that are required for your web application. Expand the **Select the Dojo components to be included in the project** section and select the Dojo components that you want to include in your project. **Tip:** To select one of the most commonly selected configurations, choose an item from the **Common Configurations** list:

- **Minimal** includes `dojo/dojo.js`, `dojo/_base.js`, and `dojo/_base`.

- **Dojo Core** includes `dojo`.

- **Dijit** includes `dijit` and `dojo`.

- **Dojox Mobile** includes `dijit`, `dojo`, `dojox/fx`, `dojox/mobiledojox/fx.js`, and `dojox/mobile.js`.

To include Dojo support in your web project, select `dojo/dojo.js`, `dojo/_base.js`, and `dojo/_base` (the minimal configuration).

Note: The test code included in Dojo Toolkit SDK 1.6 and previous versions is vulnerable to cross-site scripting attacks. These vulnerabilities are contained in several `.php` files that supply dummy data for the test cases. When you select these files to copy into the project, a warning message is displayed in the Dojo Project Setup page of the wizard. Clear the selections for these files so that they are not copied into the project. If you require these files, ensure that they are not deployed with your web application.

11. Click **Finish** to create your web project.

Results

In the Enterprise Explorer view, a `dojo` folder is created under the `WebContent` folder. When you create a web page, the palette opens with several Dojo drawers. The Dojo components in the drawers are supplied by IBM® at design time. At run time, the JavaScript that corresponds to the Dojo resources you specified during project creation is run.

What to do next

JavaScript files must be stored in the `WebContent` folder. JavaScript files that are referenced by HTML or JSP files run on the server even when they are not in a JavaScript source folder. A JavaScript source folder is used to determine which files to validate and make available for content assist. To set up source folders or view existing source folders in the project properties, right-click your web project and select **Properties > JavaScript > Include Path**. JavaScript files that are not referenced by HTML or JSP files must be placed in a JavaScript source folder. After these files are placed in a JavaScript source folder, they are available for validation and content assist.

- [Configuring Dojo project libraries](#)

You can configure which libraries you want your Dojo project to use.

Parent topic: [Creating Web 2.0-enabled web projects](#)

Configuring Dojo project libraries

You can configure which libraries you want your Dojo project to use.

Before you begin

[Create a Dojo enabled Web project](#)

Procedure

1. In the Enterprise Explorer view, right-click your web project and select **Properties** to display the properties window.
 2. Select **Dojo Toolkit**.
 3. To use a different Dojo toolkit:
 - A. Click **Change how this project accesses Dojo**.
 - B. Choose one of the following options to set up Dojo in your web project and click **Next**:
 - **Copy Dojo into this project. It will be deployed from there.**
 - Specify the name of the Dojo folder and which of the following Dojo Toolkits to use:
 - One that is provided with the product
 - One that is in your workspace or file system
 - You can select an archive file of a compressed Dojo distribution. When you click **Finish**, the contents of the archive file are automatically extracted into your project.
 - **Dojo is in a project in the workspace, and will be deployed from there.**
 - Browse to the root Dojo folder in another project in your workspace. The Dojo Toolkit is not copied into your project. It is deployed from the project that contains the toolkit.
 - **Dojo is remotely deployed or is on a public CDN.**
 - Specify the remote location of the Dojo toolkit source:
 - **Public CDN:** Enter the URL of a publicly available content delivery network. Content delivery networks provide geographically distributed hosting for open source JavaScript libraries. When a browser resolves the URL in your web application, the browser automatically downloads the file from the closest available server.
 - **Remote URI:** Type the URI of the remote location to the root Dojo folder.
 - In the Corresponding Dojo section, choose the Dojo source distribution that is the closest match to your remote Dojo Toolkit. This selection provides access to tools such as content assist. You can choose the default Dojo provided with this product or browse to a Dojo folder in your workspace or file system.
 - C. If you copied the Dojo Toolkit into your project, you can include only the parts of the Dojo Toolkit that are required for your web application. Expand the **Select the Dojo components to be included in the project** section and select the Dojo components that you want to include in your project. **Tip:** To select one of the most commonly selected configurations, choose an item from the **Common Configurations** list:
 - **Minimal** includes `dojo/dojo.js`, `dojo/_base.js`, and `dojo/_base`.
 - **Dojo Core** includes `dojo`.
 - **Dijit** includes `dijit` and `dojo`.
 - **Dojox Mobile** includes `dijit`, `dojo`, `dojox/fx`, `dojox/mobiledojox/fx.js`, and `dojox/mobile.js`.
 - To include Dojo support in your web project, select `dojo/dojo.js`, `dojo/_base.js`, and `dojo/_base` (the minimal configuration).
- Note:** The test code included in Dojo Toolkit SDK 1.6 and previous versions is vulnerable to cross-site scripting attacks. These vulnerabilities are contained in several .php files that supply dummy data for the test cases. When you select these files to copy into the project, a warning message is displayed in the Dojo Project Setup page of the wizard. Clear the selections for these files so that they are not copied into the project. If you require these files, ensure that they are not deployed with your web application.

4. Click **Finish**.

5. Modify the paths to the Dojo root, loader, or CSS files and click **OK**.

Results

When you create a new web page and add Dojo widgets to the page, the required include statements that are generated into the <head> tag of the page by using the path that is specified in the project properties.

Parent topic: [Creating Dojo-enabled web projects](#)

Web 2.0 features

The Web 2.0 features define the characteristics of your Web 2.0-enabled web application. These features also specify the requirements and constraints that apply to your Web 2.0 project.

Table 1. Web 2.0 features

Feature name	Description
Web 2.0	Provides support for the runtime components in the WebSphere® Feature Pack for Web 2.0: Ajax Proxy, Dojo Toolkit, and Server-side technologies.
Ajax Proxy	Provides support for the WebSphere Ajax Proxy. By adding the Ajax Proxy feature, your web project is configured to broker client requests from multiple domains while using Ajax. The Ajax proxy included in WebSphere Application Server Feature Pack for Web 2.0 eliminates browser security concerns with cross-domain scripting when combining internal and external services.
Dojo Toolkit	Provides Dojo capabilities. If the Dojo Toolkit feature is added to your web project, you can develop Dojo web applications. The Dojo Toolkit included in WebSphere Application Server Feature Pack for Web 2.0 includes these components: The open source Dojo Toolkit . Additional IBM® extensions to the base Dojo Toolkit. For example, libraries for Atom (Atom Syndication Format) data access, analog and bar gauges, and simplified access for SOAP web services.
Server-side technologies	Adds the libraries for server-side technologies such as Feed support, web remoting, web messaging, and JSON4J to your web project. Tip: This product supports the server-side technologies that are included in WebSphere Application Server Feature Pack for Web 2.0 and Mobile, and includes the following feature versions: Server-side technologies feature version 1.1.0 for the Feature pack for Web 2.0 version 1.1.0. Server-side technologies feature version 1.0.0.1 for the Feature pack for Web 2.0 version 1.0.0.1, 1.0.1, and 1.0.1.1. Server-side technologies feature version 1.0 for the Feature pack for Web 2.0 versions 1.0, 1.0.0.1, 1.0.1, and 1.0.1.1. If the Server-side technologies feature is added to your web project, you get development and runtime support for WebSphere Application Server Feature Pack for Web 2.0 . All of the required JAR files are added to the Java™ Build Path for your project and the deployment descriptor.

Parent topic: [Creating Web 2.0-enabled web projects](#)

Web project features

A project feature is a specific unit of function that you can add to a project when that function is required. When you add a project feature to a project, it can add natures, builders, class path entries, and resources, depending on the characteristics of the project. Features define characteristics and requirements for Java™ EE projects and are used as part of the runtime configuration.

When you add a feature to a project, that project is configured to perform a certain task, fulfill specific requirements, or have certain characteristics. For example, you can use the EAR feature to set up a project to function as an enterprise application. The EAR feature adds a deployment descriptor and sets up the class path for the project.

You can add features only to Java EE projects and other types of projects that are based on Java EE projects. These projects can include enterprise application projects, web projects, and EJB projects. For example, you cannot add features to a Java project or plug-in project. Typically, a feature-enabled project has at least one feature when it is created; you can add more features if required. For example, a new EJB project has the EJB Module feature. You can then add other features to this project like the EJBDoclet (XDoclet) feature.

Some features require other features as prerequisites. Other features cannot be in the same project together. For example, you cannot add the Dynamic Web Module feature to an EJB project because the EJB project already has the EJB Module feature. Some features can be removed from a project and others cannot.

Table 1. Features used in web application development

Project feature	Description	Dependencies
Default style sheet (CSS file)	Adds an automatically generated CSS file to your project.	Requires one of the following web modules:Dynamic Web Module v2.2+Static Web Module
Default synchronization policy for CVS repository 1.0	Generates a .cvsignore file for the classes directory under the WEB-INF directory.	Dynamic Web Module v2.2+
Design Time Page Template support	Helps you to create HTML and JSP files with a consistent appearance by using design-time page templates.	Requires one of the following web modules:Dynamic Web Module v2.2+Static Web Module
Dynamic Page Template support (Tiles)	Adds support for Tiles to dynamic page templates.	Dynamic Web Module v2.2+
Dynamic Web Module	Adds support for the Java Servlet API, for generation of dynamic web page content.	Dynamic Web Module v3.0 depends onJava v1.6+Dynamic Web Module v2.5 depends on Java v1.5+Dynamic Web Module v2.4, 2.3, 2.2 depend on Java v1.3+
Java	Adds support for writing applications using Java programming language.	
JavaScript Toolkit	Adds advanced JavaScript editing support.	
JAX-RS (REST Web Services)	Helps you to create Representational State Transfer (REST) services.	Dynamic Web Module v2.3+Java v1.5+

JSR Portlets on WebSphere® Portal	Adds support for JSR Portlets running on WebSphere Portal Server.	
JPA	Adds support for writing persistent metadata using the Java Persistence API (JPA).	Java v1.5+
JSTL	The JSP Standard Tag Library. Important: This product supports JSTL 1.2 in web projects. Since WebSphere Application Server Version 7.0 supports JSTL 1.0, 1.1, and 1.2, you do not need to include the JSTL libraries in the web application project. To ensure that the web project contains the JSTL functions, select JSTL v1.1.	JSTL v1.1 depends on Dynamic Web Module v2.3+JSTL v1.0 depends on Dynamic Web Module v2.2-2.3
SIP Module	Adds support for Session Initiation Protocol (SIP) projects.	SIP v1.1 depends on Dynamic Web Module v2.5SIP v1.0 depends on Dynamic Web Module v2.3-2.4
Static Web Module	Adds support for Static web projects.	
Web 2.0	Adds support for the runtime components in the WebSphere Feature Pack for Web 2.0: Ajax Proxy, Dojo Toolkit, and Server-side technologies.	
Ajax Proxy	Adds support for the WebSphere Ajax Proxy. By supporting the Ajax proxy feature, your web project is configured to broker client requests from multiple domains while using Ajax. The Ajax proxy included in the WebSphere Application Server Feature Pack for Web 2.0 eliminates browser security concerns with cross-domain scripting when combining internal and external services.	Dynamic Web Module v2.2+

Dojo Toolkit	Adds Dojo capabilities. By supporting the Dojo Toolkit feature, your web project is configured to develop Dojo web applications. The Dojo Toolkit included in the WebSphere Application Server Feature Pack for Web 2.0 includes the open source Dojo toolkit and additional IBM® extensions to the base Dojo Toolkit. It includes libraries for Atom (Atom Syndication Format) data access, analog and bar gauges, and simplified access for SOAP web services.	Requires one of the following web modules: Dynamic Web Module v2.2+ Static Web Module
Server-side technologies	Adds the libraries for server-side technologies such as Feed support, web remoting, web messaging, and JSON4J to your web project. By supporting the server-side technologies feature, your web project is configured for both development and runtime support for the WebSphere Application Server Feature Pack for Web 2.0 . All of the required JAR files are added to the Java Build Path for your project and the deployment descriptor.	Dynamic Web Module v2.2+
Web Fragment Module	Adds support for the Web Fragments, which are used with web projects for generation of dynamic web page content.	Java v1.6+
WebSphere SAML Support 1.0	Adds support for WebSphere SAML libraries.	
WebDoclet (XDoclet) 1.2.3	Adds support for the project to run Weboclet post-processing on annotated servlets.	Dynamic Web Module v2.2+

WebSphere Web (Co-existence)	Adds support to deploy the project to a WebSphere server, without preventing it from being deployed to other servers.	WebSphere Web (Co-existence) v8.0 depends on: Dynamic Web Module v2.2, v2.3, v2.4, v2.5, or v3.0Java v1.3, v1.4, v1.5, or v1.6 WebSphere Web (Co-existence) v7.0 depends on: Dynamic Web Module v2.2, v2.3, v2.4, or v2.5Java v1.3, v1.4, v1.5, or v1.6 WebSphere Web (Co-existence) v6.1 depends on: Dynamic Web Module v2.2, v2.3, or v2.4Java v1.3, v1.4, or v1.5 WebSphere Web (Co-existence) v6.0 depends on: Dynamic Web Module v2.2, v2.3, or v2.4Java v1.3 or v1.4
WebSphere Web (Extended)	Helps you to deploy the project to a WebSphere server, which might prevent it from being deployed to other servers.	WebSphere Web (Co-existence) of the same version.
XML Transformations and Query 1.0	Helps you to precompile and integrate XSL style sheet documents.	Java v1.6+

Parent topic: [Creating web projects](#)

Refactoring Web project content roots

The context root is the web application root, which is the top-level directory of your application when it is deployed to the web server.

Procedure

1. Right-click your web project and select **Properties**.
2. In the Properties list, click **Web Project Settings**.
3. In the **Context root** field, type the new context root. Click **Apply**. The Rename Context Root dialog opens. Click **OK** to close the dialog.
4. Click **OK** to close the Properties dialog.
5. Expand the EAR project for your web project and then expand **META-INF**.
6. Double-click **application.xml** to open the Application Deployment Descriptor in the editor.
7. In the Application list, select the web project. In the Details section, update the **Context root** field and ensure it has the same context root that you typed in the Web Project Settings properties.
8. Save `application.xml` if needed.

Parent topic: [Creating web projects](#)

Creating website styles

Websites and web pages are a part of every web application. Each web page helps to achieve the overall goal of the entire website through its content and design.

Before you begin

[Create a Web project.](#)

About this task

Types of web pages range from simple HTML pages that contain no dynamic elements, to advanced Java™ pages that use servlets, scripts, forms, or data access components, and pages that act as a web-based user interface.

A few of the many items you should consider when you design your pages are markup language, links, images, page templates, and style sheets. Some of the tools you can use to help design your web pages include Rich Page Editor, drag-and-drop components, link utilities, image-editing tools, and preview functions.

Procedure

1. Read about designing a website.
2. Create a style sheet.

What to do next

After you design and create the style for your website, you can add web pages to your website.

- [Creating style sheets](#)

You can use style sheets on your web pages to define a consistent look and feel throughout your website because you maintain the contents (web pages) and the design (the style sheet) separately.

Parent topic: [Developing web applications](#)

Creating style sheets

You can use style sheets on your web pages to define a consistent look and feel throughout your website because you maintain the contents (web pages) and the design (the style sheet) separately.

Procedure

1. From the Web perspective, select **File > New > CSS File** and then select **Next**.
2. Select the appropriate container for the file from the list of project folders (and subfolders). You might want to create a Theme folder in which to place the CSS file under the WebContents folder. Whichever folder you choose, make sure that it is located under the WebContents folder of the web project. Choosing a folder in that location ensures that the file is available for validation, publishing, and deployment.
3. Type a file name into the appropriate field. The .css file type is appended when the file is created.
4. Click **Finish** to create and open the file in the editor.

- **Specifying styles for predefined areas**

You can specify a style for a particular area in your page. The style is defined by the *style* attribute of an HTML tag, rather than by being specified between `<STYLE>` and `</STYLE>` tags.

- **Setting the CSS profile**


You can set the level of CSS to use for your web pages. The level of CSS determines the selectors and properties that you can use in the style sheet.

Parent topic: [Creating website styles](#)

Specifying styles for predefined areas

You can specify a style for a particular area in your page. The style is defined by the *style* attribute of an HTML tag, rather than by being specified between `<STYLE>` and `</STYLE>` tags.

Procedure

1. In the Design or Split pane of Rich Page Editor, select the area where you want to specify a style.
2. In the Text tab of the Properties view, click **Browse** next to the **Properties** field. Properties: 
3. Add a specific style definition using the New Style dialog box.
4. Click **OK** to save changes and apply the class to the selected text. The style definition is embedded in the HTML tag.
For example, `<p style='color: #ff8000; font-style: oblique;>`. **Note:** If embedded styles are in the HTML file, the embedded styles override the style specifications in an external style sheet.

Parent topic: [Creating style sheets](#)

Setting the CSS profile

You can set the level of CSS to use for your web pages. The level of CSS determines the selectors and properties that you can use in the style sheet.

Procedure

1. Right-click the project for which you want to set the CSS profile. Select **Properties**.
2. Click **Web Content Settings**.
3. In the **CSS profile** field, select the profile that you want to use from the drop-down list. The following World Wide Web Consortium (W3C) cascading style sheet standards are supported:
 - CSS1 (cascading style sheet level 1)
 - CSS2 (cascading style sheet level 2)
 - CSS3 (cascading style sheet level 3)
 - CSS Mobile Profile 1.0
 - WCSS 1.0
4. Click **Apply** and then **OK** to close the Project Properties page.

Results

Now that the CSS profile for your web project is set, open the style sheet in the editor. Press **Ctrl + Space**. The selectors and properties that appear differ depending on what level of CSS you are using. For example, if you set the CSS profile to CSS1, the following proposals are displayed: @IMPORT, A, ABBR, and so on. If you set the CSS profile to CSS2, a different content model is displayed with the following proposals: @IMPORT, @CHARSET, @MEDIA, @PAGE, and so on.

Parent topic: [Creating style sheets](#)

Creating and importing web pages and resources

After you create a web project, you can populate your web application with the required web pages and web resources that complement the high-level goal and supporting goals of your web application.

- **Creating web pages**

The New Web Page wizard helps you to create HTML and Dojo Mobile HTML.

- **Creating servlets**

You can use the servlet wizard to create Java™ servlets. The wizard walks you through the creation process and provides output files for use with your web application. The servlets can run on WebSphere® Application Server or other Java EE-compliant web servers.

- **Creating filter classes**

The Filter wizard enables you to create Java filter classes for various Java EE filter types, such as authentication filters, encryption filters, and data compression filters.

- **Creating listener classes**

The Listener wizard enables you to create Java listener classes for various Java EE listener types that are related to servlet context and session events and attributes.

- **Importing web archive (WAR) files**

You can use the WAR Import Wizard to import WAR files into your project.

- **Exporting web archive (WAR) files**

A web archive (WAR) file is a packaged web application that can be exported to test, publish, and deploy the resources that are developed within a web project. You can export a WAR file from a web project.

Parent topic: [Developing web applications](#)

Related tasks:

[Creating web projects](#)

Creating web pages

The New Web Page wizard helps you to create HTML and Dojo Mobile HTML.

Procedure

1. Right-click on **WebContent** in the Project Explorer
2. From the popup menu, click **New** > **Web Page**. The New Web Page wizard displays.
3. Select a template from the Template dialog box, and then type a name for the file. If you do not specify a file extension, it is added automatically.
4. Optionally change the folder in which you want to create the new web page. If you change the folder, make sure that you choose one in or under the WebContent folder; Changing the folder ensures that the file is available for validation, publishing, and deployment.
5. Check or clear the **Link page to template** box. If this box is checked, changes that you make to the template is reflected in all the web pages you create that are linked to the template. In most instances, you probably want to leave this box checked.
6. To change the options for the web page template you selected, click **Options**. The options that appear differ based on the type of web page you are creating or on the type of facets that are associated with the project.**Note:** You can select facets when you create a project, or add or remove them after the project is created by right-clicking the project and selecting **Project** > **Properties** > **Project Facets** .

Options include:

- Document Markup

- Specify encoding, content type, and doctype for the generated page. If you want to create an HTML 5 page, in the **Document Type** field, select **HTML 5**.

- Dojo Toolkit

- Enable the use of Dojo technology.

- JavaScript Files

- Specify JavaScript files that you want to add to the page.

- Mobile Web Page

- Specify the style sheet options that you want to add to the page.

- Style Sheets

- Specify style sheets that you want to add to the page.

- Adding elements to a web page from the palette

You can populate your pages with content by dragging and dropping elements from the palette to the web page.

- Adding Flash objects to a web page

You can populate your pages with Flash objects by dragging and dropping from the palette to the web page.

- JavaServer Pages (JSP) technology

- Defining HTML file preferences

- Specifying the document type declaration (DOCTYPE) for web pages

Parent topic: [Creating and importing web pages and resources](#)

Adding elements to a web page from the palette

You can populate your pages with content by dragging and dropping elements from the palette to the web page.

Before you begin

1. [Create a web project.](#)
2. [Create a web page.](#)

Procedure

1. In the Enterprise Explorer, double-click the web page to open the file in the editor.
2. Add various elements to your web page by dragging and dropping objects from the different drawers in the Palette, for example radio buttons, check boxes, or submit buttons. In the Web perspective, the Palette is by default located in the workbench, underneath the Outline and Snippets view.
3. You can select multiple elements by pressing the **CTRL** key and then perform actions on the selected elements from the context menu, for example copy, paste, or delete actions.
4. When you have finished adding elements to your web page, press **CTRL+S** to save your changes.

Parent topic: [Creating web pages](#)

Adding Flash objects to a web page

You can populate your pages with Flash objects by dragging and dropping from the palette to the web page.

Before you begin

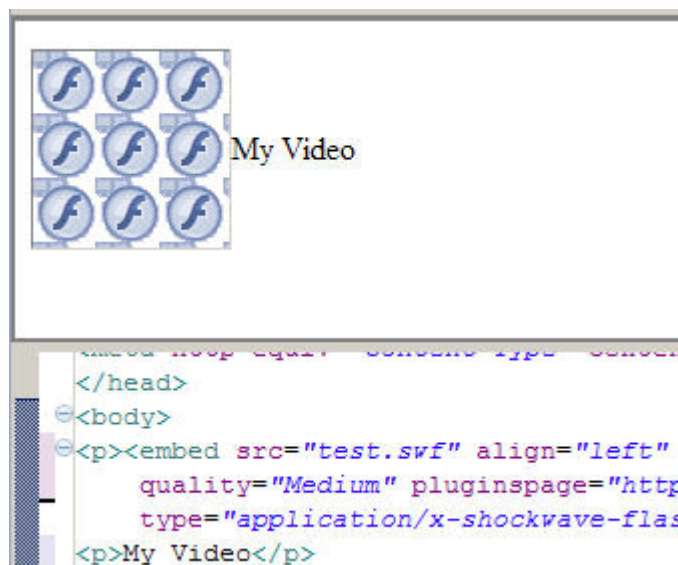
1. [Create a web project.](#)
2. [Create a web page.](#)

Procedure

1. In the Enterprise Explorer, double-click the web page to open the file in Rich Page Editor.
2. Open the HTML Tags drawer in the palette. In the Web perspective, the palette is by default located underneath the Outline and Snippets view. If the palette is not shown, click the Show/Hide Palette icon in the editor toolbar.
3. Drag the Flash element to your web page. The Embed Flash Plugin window opens.
4. In the **Flash location** field, type or browse to the location of the Flash file, for example an *.flv or *.swf file.
5. When you have finished adding Flash objects to your Web page, press **CTRL+S** to save your changes.

Results

A Flash icon appears on your web page in the Design pane or Split pane of the editor.



Parent topic: [Creating web pages](#)

JavaServer Pages (JSP) technology

JavaServer Pages technology enables you to generate dynamic web content, such as HTML, DHTML, XHTML, and XML files, to include in a web application. JSP files are one way that the product implements server-side dynamic page content. JSP files enable a web server, such as WebSphere® Application Server or Apache Tomcat, to add content dynamically to your HTML pages before they are sent to a requesting browser.

When you deploy a JSP file to a web server that provides a servlet engine, it is preprocessed into a servlet that runs on the web server. This is in contrast with client-side JavaScript (within `<SCRIPT>` tags), which is run in a browser. A JSP page is ideal for tasks that are better suited to execution on the server, such as accessing databases or calling Enterprise beans. You can create and edit a JSP file in the HTML editor by adding your own text and images by using HTML, JSP tagging, or JavaScript, including Java™ source code inside of scriptlet tags. Typically, JSP files have the file extension `.jsp`. Additionally, the JSP specification suggests that JSP fragment files have file extension `.jspx`. If this convention is not followed, the JSP validator treats JSP fragments as regular stand-alone JSP files, and compilation errors might be reported.

You can create custom JSP tags. Custom tags simplify complex actions and provide developers with greater control over page content. Custom tags are collected into a library (taglib). A tag library descriptor file (taglib.tld) is an XML document that provides information about the tag library, including the taglib short name, library description, and tag descriptions. To use custom taglibs, you can import the tag library.tld and .jar files into your project to use them, or associate them as web Library projects. You can also reference a TLD file by using a URI.

Parent topic: [Creating web pages](#)

Defining HTML file preferences

Procedure

1. From the **Window** menu, select **Preferences**.
2. In the Preferences window, select **Web > HTML Files**.
3. Configure the HTML file preferences.
4. Click **Apply** then **OK** to save your changes.

Parent topic: [Creating web pages](#)

Specifying the document type declaration (DOCTYPE) for web pages

About this task

An implicit document type is used for content assist (Ctrl+Spacebar), the Properties view, and other editing contexts when no DOCTYPE declaration is specified in an HTML, XHTML, or JSP file. Typically, this feature is used for a fragment (partial document) that can be included in other pages by using JSP include or server-side include.

To assign an implicit document type to an HTML, XHTML, or JSP fragment:

Procedure

1. From the Enterprise Explorer view, select the HTML, XHTML, or JSP fragment.
2. Select **Properties** from the file's menu.
3. In the **Web Content Settings** page, select the wanted DOCTYPE from the **Document type** list. The **Properties** dialog for web projects also includes a **Web Content Settings** page, where you can specify project-default HTML document type. This setting is used if no DOCTYPE declaration is specified in the file, and no default document type is specified in the **Web Content Settings** page on the **Properties** dialog for the file.

Parent topic: [Creating web pages](#)

Browser requirements for Rich Page Editor

Rich Page Editor uses embedded browsers to produce a visual representation of a web page in the Design view. The browsers that are available in Rich Page Editor and their installation requirements vary according to the platform.

Procedure

The following table lists and describes the supported browsers in Rich Page Editor, by platform:

Platform	Supported browsers
Windows	Internet Explorer Available for all installations; uses the native browser code in Windows. Firefox Firefox support for Windows is embedded in the product and is functionally equivalent to a Firefox version 3.6 installation. Firefox is available only on 32-bit installations of the product. Safari Safari for Windows can be installed separately. After installation Rich Page Editor can be used in Safari. Safari support is only available on 32-bit installations of the product.
Linux	Firefox or WebKit The product attempts to locate and use browser code: WebKitGTK+librariesXULRunner installation The editor operates with a compatible XULRunner installation that is in the range of Firefox version 3.0 to version 3.6. You can also use WebKitGTK+ libraries with some additional setup. The Firefox indicators are still used in the editor even if you create a webkit-based browser. For more information about setting up the Linux browser, see Embedded browsers for Linux .
Mac	Safari The native Safari browser is automatically used for products that are available on the Mac platform.

The supported browsers are available from the editor toolbar in both the design and split views.

On the toolbar, click the icon for the browser you want to use. For example, in the following screen capture, Firefox, Internet Explorer, and Safari are supported.



Embedded browsers for Linux

On Linux systems, to ensure that product features, such as the Rich Page Editor use an appropriate embedded web browser, additional steps to configure the browser are necessary.

Product features that use an embedded web browser might not work correctly if an inappropriate browser is used. Using an inappropriate browser can cause problems such as: scenarios that fail, error messages, or an unexpected output. Product features that use an embedded browser include:

- Rich Page Editor
- Web Browser component
- Welcome page

The Eclipse Standard Widget Toolkit (SWT) supports the following browser types for Linux systems:

- Mozilla (Firefox) through the XULRunner package
- WebKit through WebKitGTK+ shared libraries

The version of Eclipse included in the product determines the default browser type used by SWT. However, you can explicitly configure the default browser type. Only one browser type is available at a time within the product.

- For Eclipse versions 3.7 and later, the WebKit browser is the default browser on Linux. If suitable WebKit libraries are not found, the XULRunner browser is used.

Configuring for the WebKit embedded browser

A WebKit embedded browser is supplied as a separate installation of the WebKitGTK+ shared libraries, however; these libraries are included in many of the supported Linux distributions.

Procedure

If necessary, install the WebKitGTK+ package onto the system and ensure that it is included on the default library path.

Configuring for the XULRunner embedded browser

The XULRunner package enables Mozilla as the embedded browser. If several XULRunner packages are installed on the same system, version mismatches can occur even if a specific XULRunner installation is registered as the default version. To clearly define the XULRunner browser and level to be used in your configuration, you must set up an explicit pointer to a XULRunner version.

About this task

The supported XULRunner versions are:

- 1.8.x
- 1.9.2
- 3.6.x

Note: The XULRunner package must match the architecture (32-bit or 64-bit) of the product installation.

To download the XULRunner 1.9.2, click one of the following links:

- [XULRunner 32-bit download](#)
- [XULRunner 64-bit download](#)

Procedure

To set up an explicit pointer to a XULRunner version, complete the following steps.

1. In the `eclipse.ini` file included in the product installation, locate the `-vmargs` section. **Note:** For users of IBM MobileFirst™ Platform Foundation only:
 - If a `worklight.sh` file is present in the same product directory as the `eclipse.ini` file, add your updates to the `-vmargs` sections of both files.
 - Some installations use JRE arguments from the `worklight.sh` script instead of from the `eclipse.ini` file.
2. In the `-vmargs` section, add the following JVM system variable where `/home/myuser/xulrunner` is the path to the root of an uncompressed XULRunner package.
`-Dorg.eclipse.swt.browser.XULRunnerPath=/home/myuser/xulrunner`

Complete the following step to use the XULRunner browser instead of the WebKit browser.

1. Add the following JVM parameter to the `-vmargs` section at the end of the `eclipse.ini` file. **Note:** For users of MobileFirst Studio only: If the `worklight.sh` file is present, add the same code to the end of this file.

```
-Dorg.eclipse.swt.browser.DefaultType=mozilla
```

Setting the Rich Page Editor preferences

You can customize the display of Rich Page Editor by setting the preferences for view shortcuts, pane visibility and layout, design mode, and web browser.

Procedure

1. In the main menu, click **Window > Preferences**.
2. Expand **Web > Rich Page Editor**.
3. Specify the default preference settings for Rich Page Editor.

Editor preference	Description
View shortcuts	Specify whether to show or hide the shortcut toolbar buttons in Rich Page Editor for these views: Palette, Properties, Outline, and Mobile Views.
Visible pane	Select which view to show when you open a file with Rich Page Editor. You can choose from these views: Design, Source, and Split.
Pane layout	Set the Split view layout, which is a combination of the Source view and Design view, to split the editor view either horizontally or vertically.
Design mode	<p>Specify whether to enable or disable Design Mode. When Design Mode is available, the editing features help you to add and edit widgets in the Design view of the editor. For example, the editing features guide the placement of code when you drop a widget on a container widget. Visual cues highlight the possible drop locations and pop-up cues indicate the editing function that is available for the selected widget. Design Mode also adds dashed borders to empty table cells.</p> <p>When Design Mode is unavailable, elements in the Design view are displayed exactly as they are shown in the web browser, without any visual aids for editing.</p>

Web browser	Select the web browser in which to show the page that is being edited. Note: The list of available web browsers is dependent on the platform and web browsers that are installed on your computer.
--------------------	---

Tip: When you are working with Rich Page Editor, you can change these settings from the editor window. To change the view shortcuts, pane layout, design mode, and web browser settings, use the toolbar in the upper-right corner of the editor window. To change the visible pane, use the tabs in the lower-left corner.

4. Optional: To specify that you want to remember these preference settings for each resource, select **Remember settings for each individual resource**.
5. Specify the Smart Highlight settings for Rich Page Editor.

Smart Highlight preference	Description
jQuery	Specify whether to highlight nodes in the Design and Outline views that are matched by jQuery expression selectors in the Source view or Javascript editors. Tip: By default, matched nodes are highlighted in yellow. To change the highlight color, click Change Highlight Color .

6. Click **Apply** and then save your changes by clicking **OK**.

Opening web pages in Rich Page Editor

You can open web pages in Rich Page Editor to edit HTML files, add Dojo widgets to HTML pages, and edit web pages for mobile devices.

Before you begin

You must complete the following tasks before you can open a web page in Rich Page Editor:

1. Create a project.
2. Create a web page.

Procedure

In the Enterprise Explorer view, use one of the following methods to open a web page in Rich Page Editor:

- Double-click your web page.
- Right-click your web page and select **Open**.

Working in the Design and Split views

You can use the Design and Split views in Rich Page Editor to edit HTML files in WYSIWYG mode.

When you edit in the Design view, your work reflects the layout and style of the web pages that you build. The Design view removes the added complexity of source tag syntax, navigation, and debugging.

Use the Split view to show both the Design view and the Source view in a split screen view. Changes that you make in one part of the split screen are automatically updated in the other part. You can split the view horizontally or vertically.

About this task

The design and split views provide full access to the following features:

- Editor menu options
- Pop-up menu actions
- User interface options, such as those in the Styles view
- Drag-and-drop behavior

The Design and Split views also provide support for absolute positioning. You can see the immediate impact of design decisions more quickly than in a text editor. Using these views, you can efficiently and precisely change the composition and attributes of pages, tags, images, and effects.

Many actions available through the editor menus are also available from design element pop-up menus. To access the design element pop-up menus, select a page object, and then right-click the object.

Working in the Source view

You can use the Source view in Rich Page Editor to edit HTML and other markup text, such as embedded JavaScript. Any changes you make in the source view are also reflected in the Design and Split views.

About this task

You can also show the Source view by opening the Split view. The Split view shows both the Design and Source views, split vertically or horizontally. If you add or update an attribute value in the Source view while the Properties view is visible, the properties are also refreshed.

Table 1. Source view features

Feature	Description
Syntax highlighting	Each tag type uses different highlighting to make it easy to find a specific type of tag for editing. For example, you cannot edit read-only regions of the page which are highlighted in gray.
Unlimited undo and redo	You can incrementally undo and redo every change made to a file for the entire editing session. For text, changes are incremented one character or set of selected characters at a time.
Content assist	Content assist helps you to finish tags or lines of code, and insert macros. The available options in the content assist list are based on the tags that are defined by the tagging standard specified for the file being edited. If content assist does not automatically open, press Ctrl + Space. The content assist text is displayed in a yellow box as you type.
User-defined macros	You can access user-defined templates, which are chunks of predefined code, with content assist to help you add the tagging combinations that are used often.

Element selection	The element selection indicator is located within the vertical border in the left area of the Source view. Based on the location of your cursor, the element selection indicator highlights the line numbers that contain the elements being edited.
Pop-up menu options	You can right-click at a specific position in the editor to open the editor pop-up menu. This menu contains many of the same editing options that are available in the workbench Edit menu.
Drag-and-drop	You can drag objects from the Palette view to the position of the cursor in the Source view.
Copy and paste	You can press Ctrl + C and Ctrl + V to copy and paste a selected tag in the Source view.
Validation	You can configure an option on the preferences page to validate your code as you type: From the main menu, select Window > Preferences > General > Editors > Structured Text Editor . On the Structured Text Editor preferences page, select Report problems as you type .
Customization	You can customize the appearance of the editor on either of the following preferences pages: Window > Preferences > General > Editors > Editors (or Structured Text Editors) Window > Preferences > Web > HTML Files > Editor

The HTML 5 specification is supported only in the Source view. For example, you can use content assist to insert the `<canvas>` tag.

You can use any of the following methods to enter, insert, or delete tags and text in the Source view:

- Type the tags directly.
- Use content assist to receive prompts for valid tags.
- Select the menu items.
- Select the toolbar buttons.
- Use the Properties view to change tags.

Procedure

To edit an HTML file in the Source view:

1. Open the HTML file that you want to work with in the editor.
2. In the **Source** tab, use the available features to edit the code, as required. **Tip:** You can select attribute values, attribute-value pairs, and entire tag sets by using the double-click feature available in the editor. Use this feature to quickly update, copy, or remove content.
3. At intervals, to see the nesting hierarchies more clearly in the file, format individual elements or the entire document to restore element and attribute indentation. Right-click the editor window and select **Source > Format**.
4. Save the file.

Creating web pages in Rich Page Editor

You can create interactive web pages in Rich Page Editor.

Before you begin

Before you can create a web page in Rich Page Editor, you must create a project.

Procedure

1. Click **File > New > Web Page** to open the New Web Page wizard.
2. Specify a file name and template for the new web page, and then click **Finish**.
Your new web page opens in Rich Page Editor.

Creating web pages for mobile devices

You can create interactive web pages that are optimized for mobile devices.

Before you begin

Ensure that you complete the following tasks before you create a web page for a mobile device in Rich Page Editor:

1. Create a project.
2. Set the target device for your project.
3. Set Rich Page Editor as the default web page editor.

Procedure

1. Click **File > New > Web Page** to open the New Web Page wizard.
2. Specify a file name and choose one of the following mobile templates for the new web page:
 - **Dojo Mobile HTML template**
 - Sets up the web page for Dojo. Generates content into the web page to prepare the web page for use with Dojo libraries. This content can include:
 - JavaScript and CSS includes.
 - Basic widgets that are typically required for Dojo Mobile web pages, such as a mobile View widget.
 - **jQuery Mobile HTML template**
 - Sets up the web page for jQuery. Generates content into the web page to prepare the web page for use with its libraries. This content can include:
 - JavaScript and CSS includes.
 - Basic widgets that are typically required for jQuery Mobile web pages, such as a Page widget.
3. Optional: To open the New Web Page Options page and add more options to your mobile web page, click **Options**.

Option	Description
--------	-------------

<p>Set the document type declaration to HTML 5 and cache the page</p>	<p>From the list of options, click Document Markup. From the Document Type list, select HTML 5 to show more options. Specify the icon that is used by mobile devices when users add bookmarks. To select an icon from your workspace, click Browse next to the File href field. Enable browser application caching. In the Manifest Section field, select CACHE and then specify a manifest file. For example, <code>WebContent/META-INF/cache.mf</code>. HTML 5 application caching ensures performance and availability when the mobile device is offline. For more information about cache manifest files, see the latest HTML5 specification at: http://dev.w3.org/html5/spec, and search for "cache manifest".</p>
<p>Set the device detection and stylesheet options</p>	<p>From the list of options, click Mobile Web Page. Select one of the following options: Detect device The web page detects the device that shows the content and loads the appropriate CSS by including the script <code>dojox/mobile/deviceTheme.js</code>. Select dojox.mobile stylesheet The selected style sheet is loaded by using the <code><link></code> tag. You can select one of the following style sheets: <code>blackberry.css</code>, <code>android.css</code>, <code>ipad.css</code>, <code>iphone.css</code>. No CSS Use a style sheet other than the ones that are available when you select the <code>dojox.mobile</code> style sheet option. When you specify the <code>No CSS</code> option, you can select Stylesheets from the list of options and add the style sheets that you want to use.</p>

4. Click **Finish**. Your web page opens in Rich Page Editor.

Mobile patterns

Mobile patterns provide templates that you can use to develop pages that are associated with a jQuery or Dojo mobile application. Using mobile patterns accelerates development of your mobile application by providing views common to many mobile applications.

You can choose from many mobile patterns available in the Default Mobile Pattern Set, or you can create your own Mobile Pattern Sets. For more information, see [Creating mobile pattern projects](#).

All the available Pattern Sets in your workspace and the Default Mobile Pattern Set appear grouped in the Pattern Set combination box. You can select any Mobile Pattern Set and see its content on the Add Mobile Page window.

Each Pattern Set contains categories and each category groups a list of patterns, for example: The Default Mobile Pattern Set are grouped into four categories.

Selecting a category on the Add Mobile Page window displays a list of available patterns that are associated with the category.

- Lists

- Choose from a number of different list formats from simple to complex. You can choose unordered lists patterns or ordered list patterns.

- Authentications

- Choose the type of login page for your application that contains only a User ID and password fields. Or, select a template that contains more input areas or buttons, such as **forgot password** and **register**.

- Navigation and search

- Choose from various navigation patterns, which include toolbars, navigation lists, or lists with searchable content.

- Configuration

- Choose from blank configuration pages to pages that contain predefined configuration items, such as language.

Some mobile patterns are sets where mobile views within the set are appropriately linked. For example, selecting a login page with **Reset password**, the Reset password template page is also created. When you select a mobile pattern that is a set, you see all pages in the preview.

Choosing a mobile pattern adds the appropriate code into your application after which you can alter it as required.

Related tasks:

[Adding a mobile pattern to an application](#)

Adding a mobile pattern to an application

Use mobile patterns to accelerate development of mobile applications. Select from a predefined list of mobile patterns to quickly add code to your application.

Before you begin

If the Mobile Navigation View is not shown, go to **Window > Show View > Other > Web > Mobile Navigation** to display it.

Procedure

1. In the Mobile Navigation View, click the plus sign icon.
2. In the add window, select a category and click **Create view from UI pattern**. The available patterns that are associated with the category are loaded in the view.
3. Required: Select the mobile pattern and click **Finish** to insert into your application.

Related concepts:

[Mobile patterns](#)

Creating mobile pattern projects

The UI Pattern is a container for mobile patterns. You can add mobile patterns to either a Dojo or jQuery app. You can also add your own mobile patterns into the tool.

Procedure

1. Use the UI Pattern Project wizard to create your own pattern project.
 - A. Click **File > New > Project**.
 - B. Expand the **Web** folder and select **UI Pattern Project**.
 - C. Click **Next**.
 - D. Give your UI Pattern Project a name.
 - E. Optional: You can click **include jQuery** and **add jQuery Resources** to the project. **Note:** Files are **jQuery Mobile** (JS and CSS files) and **jQuery JS Core** so you can properly preview the app by using the Rich Page Editor.
 - F. Click **Finish**. A UI Pattern Project is created.
2. Right-click your UI Pattern Project and select **New > UI Pattern**. **Note:** Your project must contain either Dojo framework or jQuery framework, or both, for you to be able to continue in the **UI Pattern** wizard. For instructions, see [Adding Dojo framework to a UI Pattern Project](#) and [Adding jQuery framework to a UI Pattern Project](#).
3. Define the name of your pattern and click **Finish**. A folder with the name of your new pattern is added to the `WebContent` folder. This folder contains the pattern's resources.
4. Open the `pattern.html` file that is found in one of the following locations:
 - `WebContent/pattern_name/Dojo` Or `WebContent/pattern_name/jquery`.
 - Adding widgets in the view for Dojo.
 - Adding widgets in the page for jQuery.

Note: If you are creating a Dojo pattern, ensure that the following two script tags are included in the `pattern.html` file under the `Dojo` folder.

```
<script type="text/javascript"
pattern.discardNode="true">

require([ "dojox/mobile/parser", "dojox/mobile/compat" ]);

</script>
```

This script tag is required to preview the pattern in the Mobile Pattern Browser. The `pattern.discardNode` attribute is used by patterns to identify when an element is discarded from pattern insertion.

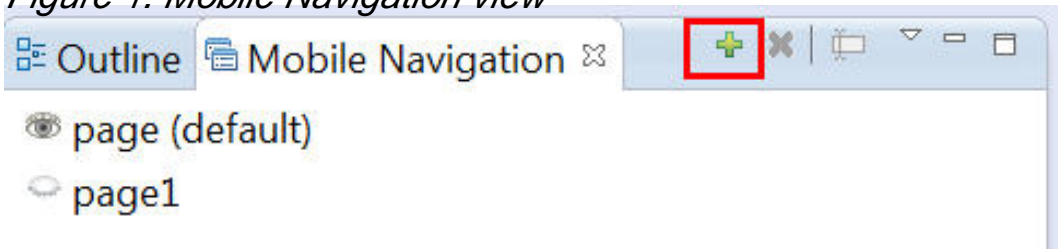
```
<script type="text/javascript">

require([ /*START_DEPENDENCIES*/ "dojox/mobile/ScrollableView" /*END_DEPENDENCIES*/ ]);

</script>
```

In the second script tag, when a pattern is added to a mobile page, the `require` elements between the `START_DEPENDENCIES` comment and the `END_DEPENDENCIES` comment are translated to Dojo module requests. Then, they are inserted into the Dojo `require` section in the final application. The `require` elements outside the `DEPENDENCIES` comments are not added to the final application. When you add a Dojo widget to your mobile pattern, add the necessary modules for that widget between the `DEPENDENCIES` comments

5. Save your UI Pattern project. You can add the mobile pattern to a Dojo or jQuery project by creating or opening a MobileFirst hybrid project with Dojo or jQuery support. Open the `index.html` file that is found under `apps / app_name / common` in Rich Page Editor.
6. Click (+) from the Mobile Navigation view. Use the following figure for guidance:
Figure 1. Mobile Navigation view



- If the Mobile Navigation view is not visible, click **Window > Show view > Other**, expand the **Web** folder, select **Mobile Navigation**, and click **OK**.
7. Complete the Add jQuery Mobile Page or Add Dojo Mobile Page wizard.
 - A. Type the name of your UI Pattern in the **ID** field.
 - B. Select the **Create from UI pattern** option.
 - C. From the **Pattern Set** menu, select **UI_Pattern_project_name**. Your mobile pattern is displayed in the Mobile Pattern Browser.
 - D. Select your mobile pattern and click **Finish**.Your new UI Pattern is displayed in the preview.
 8. Save your Dojo or jQuery project where the pattern was added.

Results

The new pattern is available for you to use with other Dojo and jQuery projects in the workspace.

What to do next

To add a mobile pattern to an application, see [Adding a mobile pattern to an application](#).

Adding Dojo framework to a UI Pattern Project

To create UI Patterns in a Pattern Project, Dojo or jQuery framework, or both, must be present in the project.

Before you begin

Before you add Dojo framework to a UI Pattern Project, a UI Pattern Project must already be created.

Procedure

1. In MobileFirst Studio, right-click your UI Pattern Project and select **Properties**.
2. In the **Properties for <Your project name>** window that appears, select **Project Facets** from the list.
3. In the **Project Facet** table, expand **Web 2.0**.
4. Select **Dojo Toolkit** and click **Apply**.
5. Click **OK**.

Adding jQuery framework to a UI Pattern Project

To create UI Patterns in a Pattern Project, Dojo or jQuery framework, or both, must be present in the project.

Before you begin

Before you add jQuery framework to a UI Pattern Project, a UI Pattern Project must already be created.

Procedure

Copy **jQuery Mobile (JavaScript, CSS, and images)** from [jQuery Mobile downloads](#) and **jQuery JS Core** from the [jQuery Core - All Versions](#) page into the **WebContent** folder of your MobileFirst Studio UI Pattern Project.

Adding elements to web pages from the palette

You can populate a web page with content by dragging elements from the Palette view to the web page in Rich Page Editor.

Before you begin

You must complete the following tasks before you can add elements to a web page in Rich Page Editor:

1. Create a project.
2. Create a web page.

Procedure

1. In the Enterprise Explorer view, double-click your web page to open it in Rich Page Editor.
2. Add various elements to your web page by dragging objects from the different drawers in the Palette view, such as radio buttons, check boxes, and submit buttons. **Tip:** In the Web perspective, the Palette view is located by default on the right side of the workbench, underneath the Outline and Snippets views.
3. You can select multiple elements by pressing Ctrl and then performing actions on the selected elements from the menu, such as copy, paste, or delete.
4. When you finish adding elements to your web page, save your changes by pressing Ctrl + S.

HTML5 tags in palette

HTML5 tags are available to use in the Rich Page Editor palette when you are creating an HTML5 web page.

The HTML5 tags can be found in the palette under the Table tag. The following tags are HTML5 tags:

- Canvas
- Audio
- Embed
- Video
- Figure
- Meter
- Progress
- Time
- Article
- Details
- Dialog
- Figcaption
- Footer
- Header
- Main
- Section

If you drag one of these widgets into the Design pane, a new tag is added to the Source pane with the correct format.

Note: This function is only visible when you are working on an HTML5 page, where these tags are legal. For example, if you create a new HTML4 web page, these HTML5 tags do not appear in the palette.

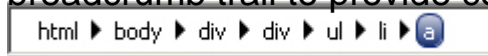
Properties view associated with Rich Page Editor

The Properties view that is associated with Rich Page Editor displays specific information for the currently selected tag in a web page. You can use the Properties view to edit properties that are related to the appearance of tags in a web browser. For example, you can change CSS style information, default attribute values, Dojo properties, and jQuery properties, as required.

You can use the Properties view to edit JavaScript, HTML, or JSP tags when the Design, Source, or Split view is open in Rich Page Editor. Changes in the Properties view are displayed in Rich Page Editor when you change the cursor focus or press Enter. If you update tags in the Source view of Rich Page Editor, your changes take effect immediately in the Properties view.

Breadcrumb navigation

When you select a node in Rich Page Editor, the Properties view uses a breadcrumb trail to provide context for the selected node:



You can scroll through the breadcrumb trail without losing the position of your cursor in Rich Page Editor. Using this feature, you can quickly update the properties of ancestor elements.

Categorized property pages

The Properties view organizes properties into various categories, including:

- Styles

- Use to manipulate basic CSS style information (such as an attribute or the class that is associated with it) or various font, color, and alignment properties.

- Layout

- Use to configure properties that control the layout of the element within the presentation of the page.

- All

- Use to view all of the attributes for an element, in a tabbed list.

- Dojo

- Use to configure Dojo-specific properties on certain widgets.**Note:** This category applies only to Dojo-enabled projects.

- jQuery

- Use to configure jQuery-specific properties on certain widgets.**Note:** This category applies only to jQuery-enabled projects.

Creating servlets

You can use the servlet wizard to create Java™ servlets. The wizard walks you through the creation process and provides output files for use with your web application. The servlets can run on WebSphere® Application Server or other Java EE-compliant web servers.

Procedure

1. In the Enterprise Explorer, expand your [web project](#).
2. Right-click the your project and select **New > Servlet** from the pop-up menu. The **Create Servlet** wizard appears.
3. Follow the project wizard prompts.

What to do next

- Modifiers

- You can select a modifier in the wizard to specify whether your servlet class is public, abstract, or final. Classes cannot be both abstract and final.

- `javax.servlet.Servlet`

- Although `javax.servlet.Servlet` is provided as the default interface, you can use the wizard to add additional interfaces to implement.

- Interface selection dialog

- This dialog is displayed if you choose to add an interface to your servlet. As you type the name of the interface, a list of available interfaces is dynamically updated to display only the interfaces that match the pattern. Choose an interface to see the qualifier and then click **OK**.

- Method stubs

- You can select any appropriate method stubs to be created in the servlet file. The stubs that are created by using the Inherited abstract methods option must be implemented if you do not intend to create an abstract servlet. This action is not required for Constructors from superclass.

- [Servlets](#)

Parent topic: [Creating and importing web pages and resources](#)

Servlets

Servlets are server-side Java™ programs that use the *Oracle Java Servlet API* and its associated classes and methods. These Java programs extend the functionality of a web server by generating dynamic content and responding to web client requests. When a browser sends a request to the server, the server can send the request information to a servlet, so that the servlet can construct the response that is sent back to the browser.

Just as applets run on a web browser and extend the browser's capabilities, servlets run on a Java-enabled web server, such as the WebSphere® Application Server and extend the server's capabilities. Because of their flexibility and scalability, servlets are commonly used to enable businesses to connect databases to the web.

Although a servlet can be a self-contained program, you can split application development into two portions:

- The business logic (content generation), which governs the relationship between input, processing, and output
- The presentation logic (content presentation, or graphic design rules), which determines how information is presented to the user

Using this paradigm, you might choose to have business logic handled by Java beans, the presentation logic that is handled by JavaServer Pages (JSP) or HTML files, and the HTTP protocol that is handled by a servlet.

You can develop, debug, and deploy servlets using the workbench. You can set breakpoints within servlet objects, and step through code to make changes that are dynamically folded into the running servlet on a running server, without having to restart each time.

Parent topic: [Creating servlets](#)

Creating filter classes

The Filter wizard enables you to create Java™ filter classes for various Java EE filter types, such as authentication filters, encryption filters, and data compression filters.

Procedure

1. Open the Filter wizard:
 - A. Open the web perspective and display the Enterprise Explorer view.
 - B. Right-click your web project and select **New > Filter**
The **Create Filter** wizard opens.
2. Specify a filter class:
 - To use an existing class, select the **Use existing Filter class** check box and then click **Browse** to locate the class.
 - To create a new filter class, specify the following information:
 - A. The **Source folder** where the filter class is placed
 - B. The **Java package** that the class belongs to (the class is added into a default package if you do not specify one)
Note: Place the listener in the Java source folder.
 - C. The **Class name** of the filter. The name that you type in the Name field is used to create a URL Mapping for the filter.
 - D. The **Superclass** for the filter class. A filter that is created by this wizard can have any class that has Object in its hierarchy as its superclass. Click **Browse** to choose from the available superclasses.
3. Click **Next**.
4. Type a description for the filter. Optionally add initialization parameters or an alternative URL mapping, and then select **Next**.
5. Select a modifier to specify whether your filter class is public, abstract, or final. (Classes cannot be both abstract and final.)
6. The javax.servlet.Filter is provided as the default **Interface**. You do not have to implement the Filter interface if you subclass a class that implements Filter, or if you implement an interface that has Filter in its hierarchy. You can also add additional interfaces to implement. Click **Add** to open the **Interface Selection** dialog. In this dialog, as you type the name of the interface that you are interested in adding in the **Choose interfaces** field, the list of available interfaces that are listed in the **Matching types** list box updates dynamically to display only the interfaces that match the pattern. Select an interface to see the **Qualifier** and click **Add**. Click **OK** when you are finished. The qualifier that you chose appears in the Interfaces dialog.
7. Choose which method stub you want to create. The **Inherited abstract methods** option adds stubs for inherited abstract methods, and that must be implemented (unless you intend to create an abstract class). Because the init(), destroy(), and doFilter() methods are all defined in the javax.servlet.Filter interface, stubs for these methods are automatically generated for each new filter class.
8. Click **Finish**.

Results

The filter that you created appears under the `Filters` icon.

Parent topic: [Creating and importing web pages and resources](#)

Creating listener classes

The Listener wizard enables you to create Java™ listener classes for various Java EE listener types that are related to servlet context and session events and attributes.

Procedure

1. Open the Web perspective and from the Enterprise Explorer, right-click on your web project and select **New > Listener**. The Create Listener wizard appears.
2. Specify a listener class. To use an existing class, select the appropriate check box and then use the **Browse** button to locate the class. Otherwise, supply the following information
 - The folder where the listener class is placed
 - The Java package that the class belongs to (it is added into a default package if you do not specify one)
 - The class name of the listener. **Note:** You should place the listener in the Java source folder.
 - The superclass for the listener class. A listener that is created by this wizard can have Object, or any class that has Object in its hierarchy, as its superclass. Click **Browse** to choose from the available superclasses.
3. Click **Next**.
4. Select the application lifecycle events to listen to. You must select at least one of the application lifecycle listeners.
5. Click **Next**.
6. Select a modifier to specify whether your listener classes are public, abstract, or final. (Classes cannot be both abstract and final.)
7. You can also add additional interfaces to implement in addition to the default interfaces. Click **Add** to open the **Interface Selection** dialog. In this dialog, as you type the name of the interface that you are interested in adding in the **Choose interfaces** field, the list of available interfaces listed in the **Matching types** list box updates dynamically to display only the interfaces that match the pattern. Choose an interface to see the **Qualifier** and click **Add**. Click **OK** when you are finished.
8. Choose which method stub you want to create. The **Inherited abstract methods** and **Constructors from superclass** options add stubs for inherited abstract methods and superclass constructors that must be implemented (unless you intend to create an abstract listener). One method with the **Inherited abstract methods** option must be created for the class not to be abstract.
9. Click **Finish**.

Results

The listener class that you created appears under the Listener icon.

Parent topic: [Creating and importing web pages and resources](#)

Importing web archive (WAR) files

You can use the WAR Import Wizard to import WAR files into your project.

About this task

A Web Archive (WAR) file is a portable, packaged Web application that you can import into your workspace.

Before importing a WAR file, you should first determine if the WAR file contains needed Java™ source files.

To import the Web project resources in a WAR file into your workspace, complete the following steps:

Procedure

1. Select **File > Import**.
2. In the Import dialog, expand the **Web** folder, select **WAR file** , and then click **Next**.
3. Locate the WAR file that you want to import using the **Browse** button.
4. The wizard assumes you want to create a new Web project with the same name as the WAR file. If you accept this choice, the project is created with the same servlet version as specified by the WAR file and in the same location.
5. Click **Finish** to populate the web project.

- **Web archive (WAR) files**

You can use the Java EE web application archive feature in this product. A web application is a group of HTML pages, JSP pages, servlets, resources, and source file, which can be managed as a single unit. A web archive (WAR) file is a packaged web application. WAR files can be used to import a web application into a web server.

Parent topic:[Creating and importing web pages and resources](#)

Web archive (WAR) files

You can use the Java™ EE web application archive feature in this product. A web application is a group of HTML pages, JSP pages, servlets, resources, and source file, which can be managed as a single unit. A web archive (WAR) file is a packaged web application. WAR files can be used to import a web application into a web server.

In addition to project resources, the WAR file includes a web deployment descriptor file. The web deployment descriptor is an XML file that contains deployment information, MIME types, session configuration details, and other settings for a web application. The web deployment descriptor file (`web.xml`) provides information about the WAR file is shared with the developers, assemblers, and deployers in a Java EE environment.

The web development environment provides facilities for importing and exporting WAR files, using the following wizards:

- Import Resources from a WAR File, which requires that you to specify a web project. You can use existing projects or create them as you use the wizard.
- Export Resources to a WAR File, which requires only an export location and some optional settings

Parent topic: [Importing web archive \(WAR\) files](#)

Exporting web archive (WAR) files

A web archive (WAR) file is a packaged web application that can be exported to test, publish, and deploy the resources that are developed within a web project. You can export a WAR file from a web project.

Procedure

1. Right-click on a web project folder and select **Export > WAR file** .
2. Specify the web project that you want to export (this field is primed if you used the popup menu to open the wizard), and specify a location for the new WAR file
3. Optional: Specify WAR export options, such as whether to include Java™ source files in the WAR, and whether to overwrite any existing resources during the export process. Source files are not included in a WAR file, because they are not necessary for the server to run the web application. You can also optimize it for a specific server runtime.
4. Click **Finish**.

Parent topic: [Creating and importing web pages and resources](#)

Creating web-based user interfaces

Learn about the different tools you can use to create your web-based user interfaces.

Procedure

- [Create Dojo applications.](#)
- [Create Web 2.0 applications.](#)

Creating Web 2.0 applications

Web 2.0 applications provide users with a richer set of controls and a more sophisticated server interaction mechanism.

About this task

Web 2.0 is the second generation of services and applications available on the World Wide Web that enable collaboration, information sharing, dynamic service delivery, and interaction. The concept of Web 2.0 typically includes later-generation web-based applications such as wikis and weblogs. Web 2.0 applications look more like desktop applications and are often dynamically data that is driven rather than comprising static HTML content.

Web 2.0 applications provide users with a richer set of controls and a more sophisticated server interaction mechanism in order to deliver web-based applications that appear like desktop applications. With Web 2.0, users do not have to refresh a page when they submit data from a browser; they can refresh only a part of page. This is achieved by using a rendering engine, such as the RPC adapter, that works from the client-side and mediates between the client and the server.

Asynchronous JavaScript and XML (AJAX) is an implementation of Web 2.0. It is made up of a group of technologies that are used to create dynamic, interactive web pages that respond quickly to requests through the exchange of smaller chunks of data. AJAX uses a combination of existing technologies and protocols including XHTML, CSS, XML, client-side scripting languages such as JavaScript, Document Object Model, and an asynchronous data retrieval mechanism such as XMLHttpRequest. It is an architecture that makes browsers more interactive by running JavaScript on the client.

To create a Web 2.0 application:

Procedure

1. [Create a Web 2.0 enabled Web project.](#)
2. Create a web page using HTML, XHTML, and CSS.
3. Create the dynamic display using Dojo widgets, JavaScript, and Document Object Model (DOM).
4. Create the data interchange mechanism using JSON, XML, and XMLHttpRequest.

What is Ajax?

Asynchronous JavaScript and XML (Ajax) refer to a group of technologies that are used to develop web applications. By combining these technologies, web pages appear more responsive since small packets of data are exchanged with the server and web pages are not reloaded each time that a user makes an input change. Ajax enables a web application user to interact with a web page without the interruption of constant web page reloading. Website interaction happens quickly with only portions of the page reloading and refreshing.

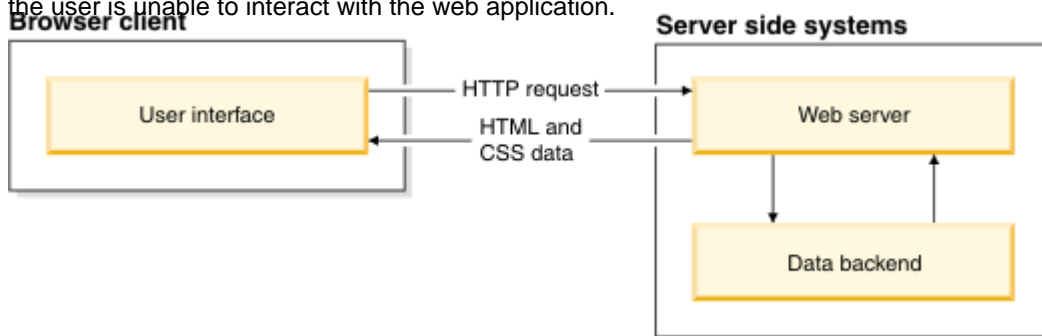
Ajax is made up of the following technologies:

- XHTML and CSS for presenting information.
- Document Object Model (DOM) for dynamically interacting with and displaying the presented information.
- XMLHttpRequest object to manipulate data asynchronously with the web server.
- XML, HTML, and XSLT for data interchange and manipulation.
- JavaScript for binding data requests and information display.

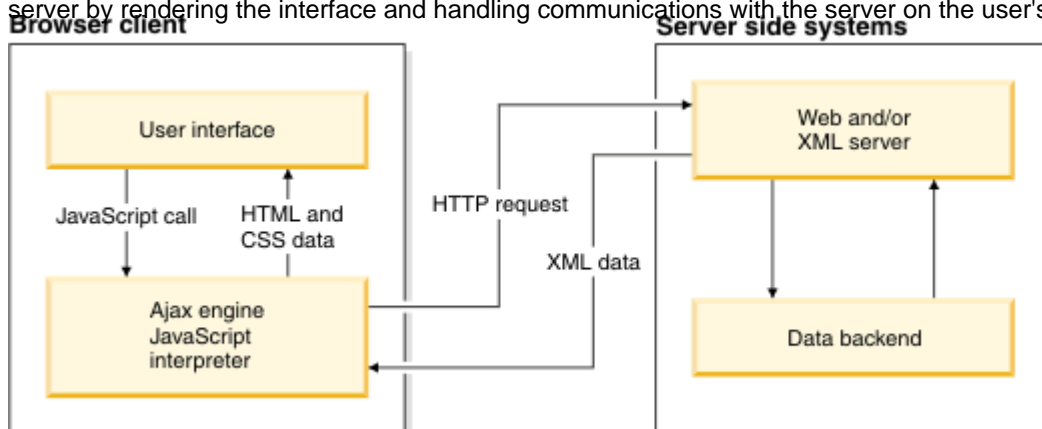
Ajax incorporates these technologies to create a new approach to developing web applications.

Ajax defines a method of initiating client to server communication without page reloads. It provides a way to enable partial page updates. From a web page user perspective, it means improved interaction with a web application, which gives the user more control of their environment, similar to that of a desktop application.

In a traditional web application, HTTP requests, that are initiated by the user's interaction with the web interface, are made to a web server. The web server processes the request and returns an HTML page to the client. During HTTP transport, the user is unable to interact with the web application.



In an Ajax web application, the user is not interrupted in interactions with the web application. The Ajax engine or JavaScript interpreter enables the user to interact with the web application independent of HTTP transport to and from the server by rendering the interface and handling communications with the server on the user's behalf.



Ajax limitations

While Ajax is a web application development technique that is designed to make web pages more responsive and interactive with a user, Ajax has some limitations to consider before you develop an Ajax-based application. The following limitations are some of the more prominent disadvantages:

- **Browser support** - Not all browsers support JavaScript or XMLHttpRequest object. Even among browsers that do have support for JavaScript and XMLHttpRequest, these objects can be treated differently. Each browser's implementation of Ajax must be considered.

- **Security and user privacy** - Not all concerns are addressed. Issues surrounding security and user privacy need to be considered when developing an Ajax application.
- **Accessibility** - Because not all browsers have JavaScript or XMLHttpRequest object support, you must ensure that you provide a way to make the web application accessible to all users.
- **Bookmark and navigation** - Since Ajax is used to asynchronously load bits of content into an existing page, some of the page information may not correspond to a newly loaded page. Browser history and bookmarks may not have the correct behavior since the URL was unchanged despite parts of the page being changed.
- **Search engine** - Ajax applications are not searchable; however, it is possible to use Ajax features and elements within an application that is searchable.

The Dojo Toolkit

The Dojo Toolkit is a powerful open source JavaScript library that can be used to create rich and varied user interfaces running within a browser. The library requires no browser-side runtime plug-in and runs natively on all major browsers. This is boon for JavaScript developers since it helps abstract away the eccentricity of different browser implementations.

The Dojo Toolkit is a powerful and flexible modular Ajax software development kit. It is broken down in to three major layers: Dojo Core, Dijit, and DojoX.

- `Dojo Core` - All the major functions that are needed to do Ajax development, plus many features that are not found in other toolkits

- `Dijit` - A high quality set of interaction rich widgets and themes for use when developing Ajax applications.

- `DojoX (Dojo eXtensions)` - A module to contain widgets and APIs that are useful for developing Ajax applications, but are not needed in all applications.

The Dojo Toolkit website serves as a guide to these layers, introducing concepts as you need them and working downward from high-level usage to building your own widgets, custom namespaces, and unit tests.

For more information about the Dojo Toolkit, refer to the <http://www.dojotoolkit.org/> website.

Creating dynamic display

Before you begin

1. [Create a Web 2.0 enabled Web project.](#)
2. [Create a Web page.](#)
3. Create server-side code to access data.

Procedure

1. [Add a Dojo widget to your Web page.](#)
2. Configure the attributes for the Dojo widgets.
3. Create the JavaScript that binds the data to the Dojo widgets.

Creating Dojo applications

The Dojo toolkit allows for the building of dynamic capabilities into web pages. You can use the components that Dojo provides to make your websites more usable, responsive, and functional.

About this task

The Dojo Toolkit is a powerful open source JavaScript™ library that can be used to create rich and varied user interfaces running within a browser. The library requires no browser-side runtime plug-in and runs natively on all major browsers. The Dojo Toolkit makes it easy to embed JavaScript controls into web pages to help improve the responsiveness and appearance of the website. Some of the benefits of adding Dojo capabilities to your applications include:

- A more interactive, differentiated experience that can lead to longer sessions and increased customer loyalty.
- Responsive, local actions that can result in fewer abandoned transactions, higher completion rates, and higher end-user productivity.

To create a Dojo application:

Procedure

1. [Create a Dojo enabled Web project](#).
2. [Create a Web page using HTML, XHTML, and CSS](#).
3. [Add Dojo widgets to the page](#) or [create custom Dojo widgets](#).
4. Create the data interchange mechanism using JSON, XML and XMLHttpRequest.
5. Enhance the performance of your Dojo application by [running a custom build](#).
6. [Test your Dojo application on the Web Preview Server](#).
7. [Debug your Web pages with Firebug](#). For more information about the Dojo Toolkit, visit the [Dojo Toolkit](#) website.


Adding Dojo widgets to web pages

You can populate your pages with Dojo widgets by dragging and dropping Dojo elements from the palette view onto the web page or by using content help on the web page Source pane.

Before you begin

1. [Create a Dojo enabled Web project.](#)
2. [Create a Web page.](#)

Procedure

1. In Enterprise Explorer, double-click the web page to open the file in the editor.
2. Open the Palette (**Window > Show View > Palette**).
3. Drag components from the different Dojo drawers in the Palette. As an example, form or layout objects.
4. When you are finished adding Dojo widgets to your web page, press CTRL+S to save your changes.
5. The Dojo widget is displayed in the Design pane. Some Dojo widgets are represented by a Dojo icon. 

Restriction: When a Dojo widget is dragged and dropped onto the Design pane, it is positioned at the side of the page because the margins are set to zero for the body element in `dojo.css`. The handle is not visible, which makes absolute positioning difficult. To work around this problem, press **Alt** and then drag the widget initially to move it. Pressing the **Alt** key while you are dragging the widget makes the absolute positioning handle visible.

Note: If Dojo is accessed from a remote URL, some Dojo widgets do not visualize properly in the Design pane, but they display in the Preview pane.

Results

In the Source view, you can see that within the `<head>` tag, script tags are added to include the Dojo bootstrap and a `dojo.requires` statement for the element, and CSS styles for the widget. The Source view supports the auto-completion of Dojo tags.

Attributes for the Dojo components can be configured in the Properties view (**Window > Show View > Properties**).

Detailed information on each type of widget is available at <http://docs.dojocampus.org>.

Creating custom Dojo widgets

You can create custom Dojo widgets in the New Dojo Widget wizard.

Before you begin

[Create a Dojo enabled Web project.](#)

About this task

Dojo widgets are composed of three files:

- An HTML file
- A JavaScript file
- A CSS file

The New Dojo Widget wizard creates these files for you. To create a custom Dojo widget:

Procedure

1. In Enterprise Explorer, right-click your Dojo enabled web project and then select **New > Dojo Widget**. Click **Next**. The New Dojo Widget wizard opens.
2. In the **Widget Name** field, type the name of the custom widget.
3. In the **Supertypes** field, type the name of the JavaScript types that the new widget extends. By default, `dijit._Widget` and `dijit._Templated` are present.
4. Click **Finish**.

Results

Your custom widget and the corresponding template and theme is created in your web project. The custom widget file opens in the editor for you to customize. The custom widget is available in the palette in the **Other Dojo Widgets** drawer.

Parent topic: [Adding support for custom tag libraries](#)

Adding Dojo widgets to existing Java EE web pages

Procedure

1. In Enterprise Explorer, right-click your web project and select **Properties**.
2. In the Properties list, select **Project Facets**.
3. In the Project Facets list, select the **Web 2.0** check box and click **OK**. A project facet is a specific unit of function that you can add to a project when that function is required. When a project facet is added to a project, it can add natures, builders, class path entries, and resources to a project, depending on the characteristics of the particular project. Web 2.0 facets define the characteristics of your web 2.0 enabled web application. The web 2.0 facets specify the requirements and constraints that apply to your web 2.0 project.

Table 1. Web 2.0 facets

Facet name	Description
Web 2.0	Enables support for the runtime components in the WebSphere® Feature Pack for web 2.0, including AJAX Proxy, Dojo Toolkit, and Server-side technologies
Dojo Toolkit	Enables Dojo capabilities.

By enabling the server-side technologies facet, your web project is configured for both development and runtime support for the [WebSphere Application Server Feature Pack for Web 2.0](#). All of the JAR files are added to the Java™ Build Path for your project and the deployment descriptor.

By enabling the Dojo Toolkit facet, your web project is configured to develop Dojo web applications. The Dojo Toolkit included in the [WebSphere Application Server Feature Pack for Web 2.0](#) includes the open source [Dojo toolkit](#) and more IBM® extensions to the base Dojo Toolkit, including libraries for ATOM (ATOM Syndication Format) data access, analog and bar gauges, and simplified access for SOAP web services.

4. [Add a Dojo widget to your Web page](#).


Editing Dojo widget attributes

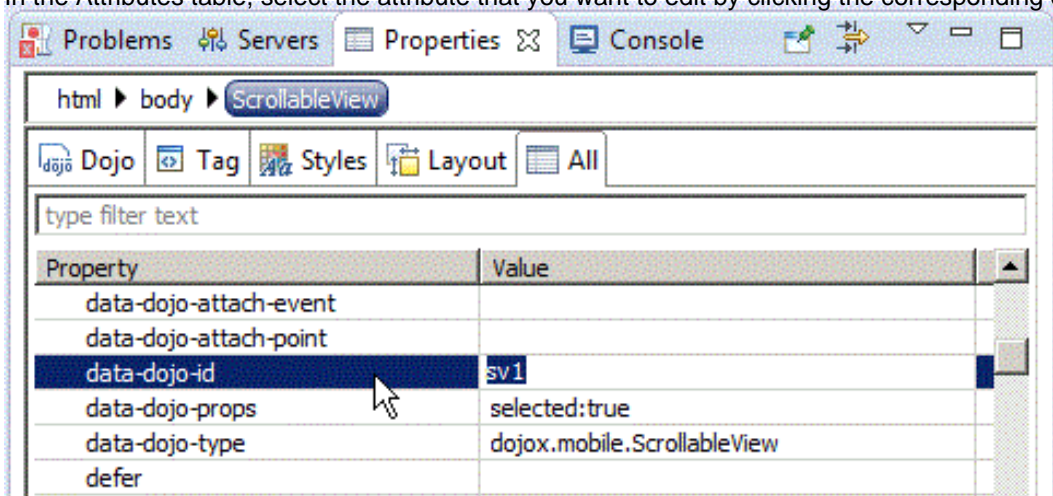
You can modify Dojo widget attributes in the Properties view.

Before you begin

1. [Create a Web 2.0 enabled Web project.](#)
2. [Create a web page.](#)
3. [Add Dojo widgets to web pages.](#)

Procedure

1. In Rich Page Editor, select the Dojo widget.
2. Open the Properties view if it is not already open (**Window > Show View > Properties**).
3. Click **All Attributes**  to display the attributes in a table view.
4. In the Attributes table, select the attribute that you want to edit by clicking the corresponding cell in the Value column:



5. Enter or select the wanted value and press CTRL+S to save your changes.

Customizing the properties for Dojo layout widgets

You can edit the attributes for Dojo layout widgets in the Properties view.

Before you begin

1. [Create a Dojo enabled Web project.](#)
2. [Create a Web page.](#)
3. Drag a Dojo layout widget from the Palette to the page.

About this task

You can also edit the child topic panes of the layout widgets.

Procedure

1. Select your widget in the Source or Design pane. The tab for your Dojo layout widget displays in the Properties view.
2. In the Properties view, in the **Style Class** field, browse to an existing style class or create a new style class.
3. In the **Properties** field, add style attributes for the tag, for example `height: 100%; width: 100%.`
4. Save your changes:

Creating Dojo classes

You can add custom Dojo classes to use in Dojo widgets.

Procedure

1. In the Enterprise Explorer view, right-click the WebContent folder in your web project and select **New > Dojo Class..**
2. In the **Dojo Class** field, enter the name of your Dojo class.
3. In the **Module Name** field, specify a module name.
4. In the **Source Folder** field, specify a folder for the class.
5. In the **Supertypes** field, specify any supertypes for the Dojo class.
6. Click **Customize Dojo Class template** to control the informational comments that are added to the class source. The Templates preferences dialog opens. Clear the check boxes of the comments that you do not want displayed in your class source. Click **OK**.
7. Click **Finish**. A JavaScript file is created with `dojo.provide` and `dojo.declare` statements. Content assist is available.

What to do next

Now you can add functions to your Dojo class, for example, constructors. The class can be instantiated in other HTML or JavaScript files in your web project.

Establishing a connection between the browser and server-side events (web messaging)

The web messaging service is a publish/subscribe implementation, which connects the browser to the WebSphere® Application Server service Integration Bus for server-side event push to the browser.

About this task

You must carefully plan prior to developing and installing a web messaging enabled application. Some items that you must plan for:

- Understand the impacts of using long-lived connections from a client to a server. Your infrastructure must support the challenges of running a web messaging enabled application.
- Determine how a browser or client connects to the web messaging service.
- The web messaging service bridges clients to the service Integration Bus for publish/subscribe operations. A service Integration Bus must be created and configured. This task requires planning.
- How messages are sent to your web messaging clients.
- Determine key metrics, such as number of concurrent clients, number of client subscriptions, publish rate. Configure a workload managed application infrastructure if a single server cannot handle scale to meet your needs.
- There are special considerations when you enable security for a web messaging enabled application.
- Understand the concerns when you deploy an application into a cluster.
- Understand what is involved in enabling an application to use the web messaging service.

Procedure

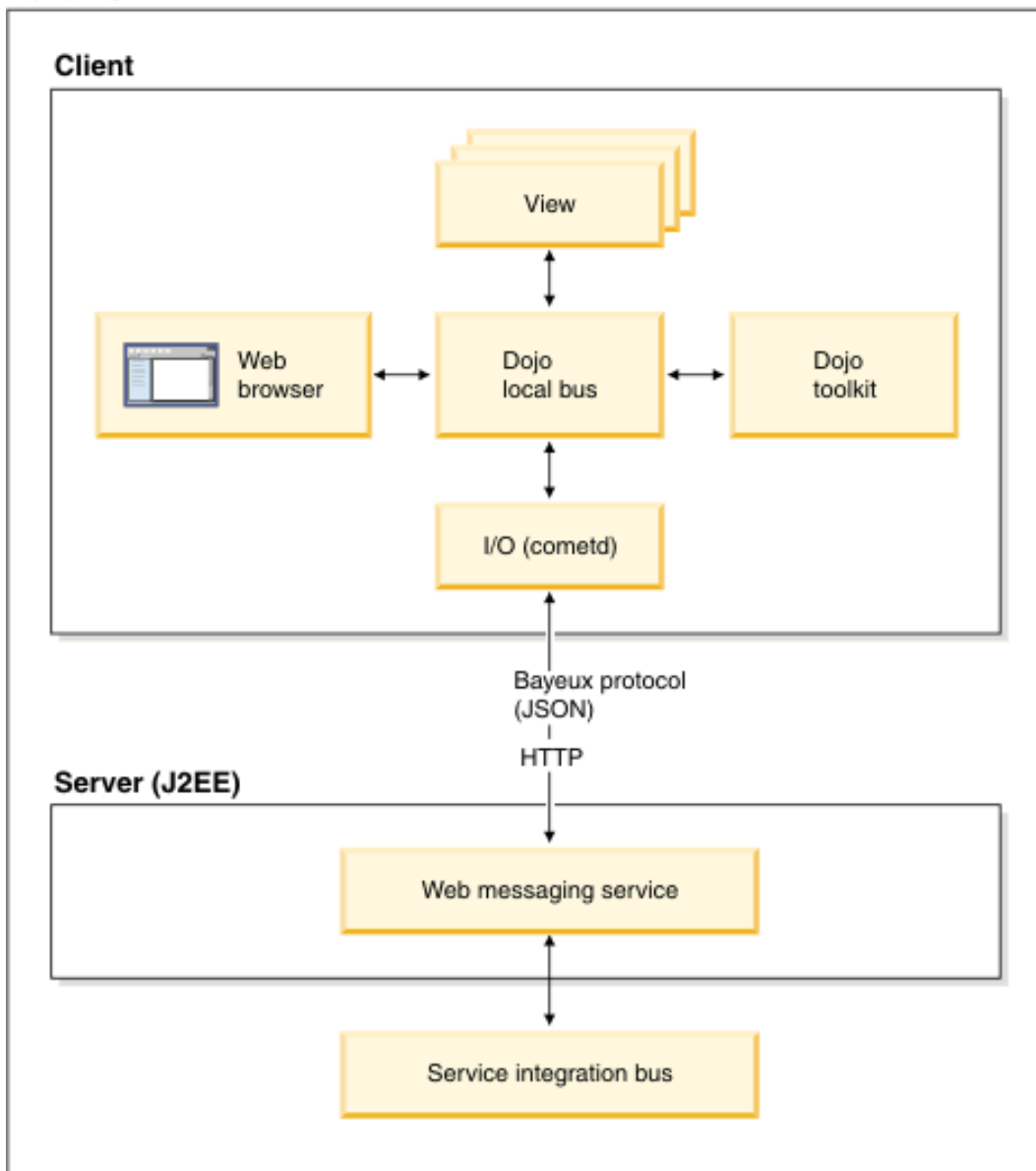
1. Plan for your web messaging enabled application. See [Getting started with the Web messaging service](#) in the WebSphere Application Server Web 2.0 and Mobile Toolkit documentation.
2. Establish a connection between the browser and server-side events. See [Using the Web messaging service](#) in the WebSphere Application Server Web 2.0 and Mobile Toolkit documentation.

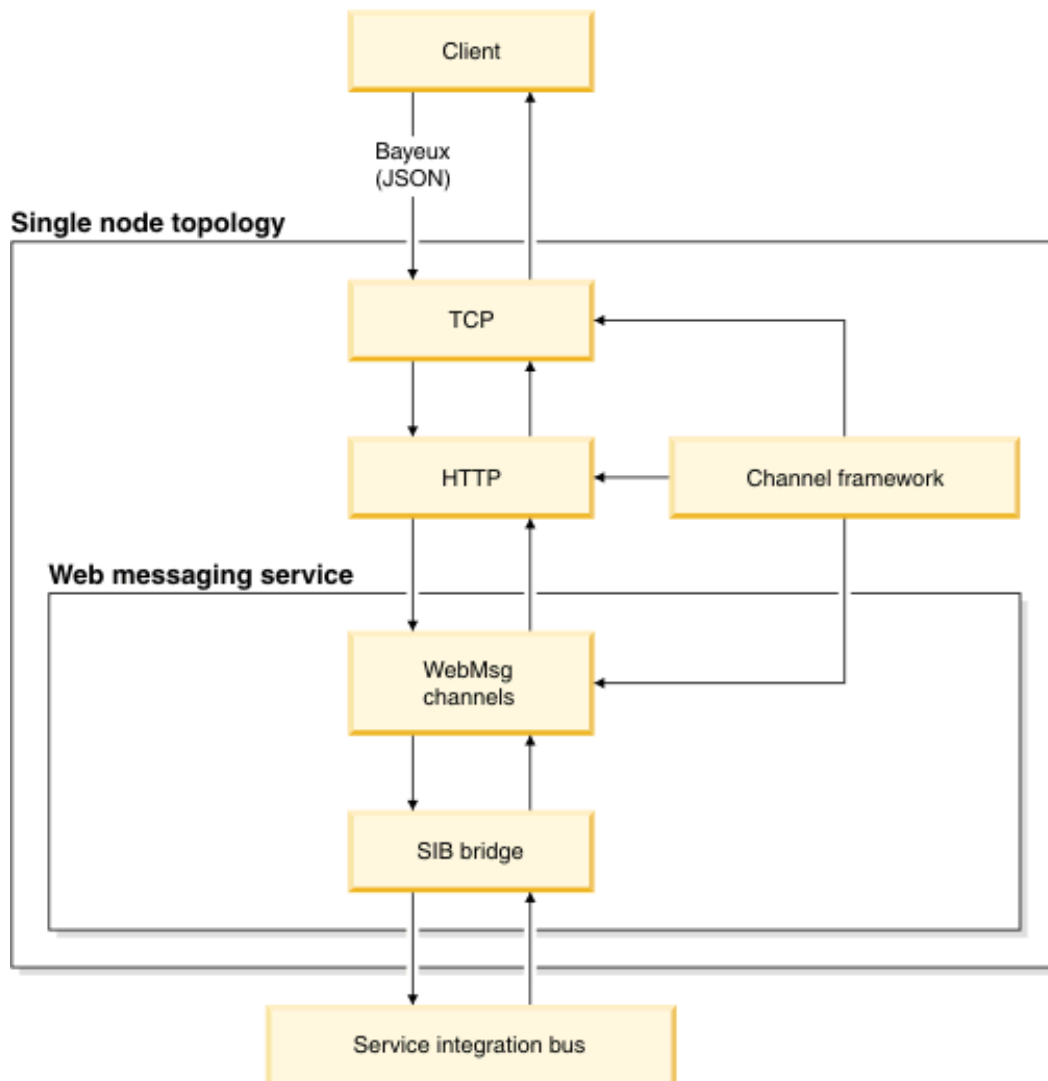
Web messaging

The web messaging service is a publish and subscribe implementation that connects the browser to the WebSphere® Application Server service Integration Bus (SIB) for server-side event push.

Client/server communication is achieved through the Bayeux protocol. The Bayeux protocol is an HTTP-based messaging routing protocol. Client support for the Bayeux protocol is provided by the Dojo Toolkit. Currently, the Dojo Toolkit is the only JavaScript library to support the Bayeux protocol, although any JavaScript library or HTTP client that implements the Bayeux protocol support can communicate with the web messaging service. The web message service implementation bridges incoming Bayeux requests to the service Integration Bus enabling web services, JMS clients, or any item that is connected to the service Integration Bus to publish events to web-based clients. You can use the web messaging service in a new or existing application by placing a run time Java™ archive (JAR) file into WebSphere Application Server, placing a utility file library JAR file in an application web module, setting up a simple configuration file, and configuring servlet mappings.

Runtime





For more information, see [Bayeux and cometd](#) website.

Scalability

In a typical application, a browser periodically requests updates from a server at a specified interval or polling. The Bayeux protocol communication types differ from the traditional polling model and communication is through a long-lived HTTP connection in which a server typically holds a connection open for a certain time to wait for an event to push to the browser. With this style of communication, the web container cannot scale as each waiting client consumes a thread that is waiting for an event. Other servers that handle this style of request have different methods for scaling. Because the web messaging service is designed to work with existing versions of WebSphere Application Server, and the current web container does not scale well with this method of communication, a new mechanism is introduced in the web messaging service to achieve scalability.

This new mechanism takes advantage of channel framework architecture and creates new channel framework channel that extends the HTTP channel to bridge incoming Bayeux requests to the service Integration Bus.

Dojo toolkit integration

Current browser support for the web messaging service is made possible with the Dojo toolkit cometd client module. Initialization, subscription, unsubscription, and publishing operations are facilitated with the cometd client. Through the cometd client, server-driven events are integrated into the Dojo event and topic system. Dojo cometd usage examples are provided in the Ajax developers guide and QuoteStreamer Sample application.

Service Integration Bus connectivity

The web messaging service connects browser clients to the built-in messaging engine and the service Integration Bus for subscribing and publishing to events and receiving messages. The underlying bridge to the service Integration Bus occurs through direct API calls to a service Integration Bus topic space. Multiple ways to publish messages to web clients exist, since the web clients are connected to the service Integration Bus. Some of these options include: standard Enterprise JavaBeans (EJB) publishing to a topic, a JMS client publishing to a topic, a web service, or the web client itself publishing to other web clients.

Rendering data as JSON

JavaScript Object Notation (JSON) enables for rapid exchange of JavaScript objects in a simple format. JSON consumes less bandwidth than XML and works well with all browsers. JSON is built from a collection of name-value pairs and ordered lists of values. The JSON4J library is an implementation of JSON for use within Java™ environments.

Procedure

- [Read about JSON4J.](#)
- [Edit JSON files with the JSON editor.](#)
- [Set JSON editor preferences.](#)

JavaScript Object Notation (JSON4J)

The JavaScript Object Notation (JSON4J) library is an implementation of a set of JavaScript Object Notation (JSON) handling classes for use within Java™ environments.

The JSON4J library provides the following functions:

- A simple Java model for constructing and manipulating data to be rendered as the JSON implementation.
- A fast transform for XML to JSON conversion, for situations where conversion from an XML reply from a web service into a JSON structure is wanted for easy use in Asynchronous JavaScript and XML (Ajax) applications. The advantage of this is that AJAX-patterned applications can handle JSON formatted data without having to rely on ActiveX objects in Microsoft Internet Explorer XML transformations and other platform-specific XML parsers. In addition, JSON-formatted data tends to be more compact and efficient to transfer.
- A JSON string and stream parser that can generate the corresponding JSONObject, which represents that JSON structure in Java. You can then change that JSONObject and serialize the changes back to the JSON implementation.

For more information about JSON, see www.json.org.

Related information:

[JSON4J usage examples](#)

Editing JSON files with the JSON editor

You can use the JSON text editor to easily modify, format, and validate your JSON files.

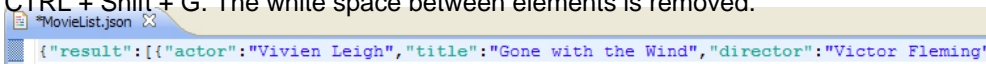
About this task

The JSON editor offers the following features:

- Syntax highlighting
- Range indication
- Source compression and formatting
- Code folding
- Bracket matching
- Syntax validation
- Outline view

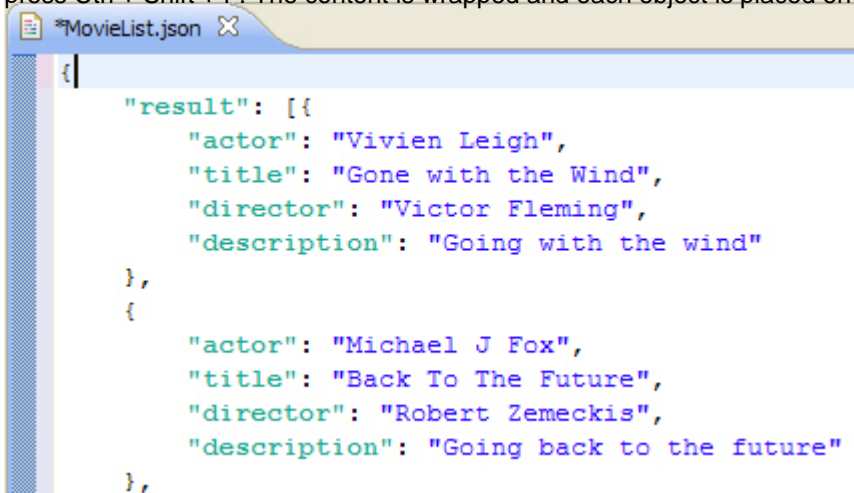
Procedure

1. In the Enterprise Explorer view, right-click your .json file or other file type that contains JSON code and select **Open With > JSON Editor**.
2. You can compress JSON strings so that the strings display on one line with white space removed between JSON elements. To compress JSON strings, right-click the editor and select **Source > Compress** from the menu or press CTRL + Shift + G. The white space between elements is removed.



The screenshot shows a code editor window titled "MovieList.json" with a single line of compressed JSON code: `{"result":[{"actor":"Vivien Leigh","title":"Gone with the Wind","director":"Victor Fleming"}`

3. To format the content so that it is more readable, right-click the editor and select **Source > Format** from the menu or press Ctrl + Shift + F. The content is wrapped and each object is placed on its own line.



The screenshot shows the same code editor window with the JSON code formatted and wrapped. The code is: `{
 "result": [
 {
 "actor": "Vivien Leigh",
 "title": "Gone with the Wind",
 "director": "Victor Fleming",
 "description": "Going with the wind"
 },
 {
 "actor": "Michael J Fox",
 "title": "Back To The Future",
 "director": "Robert Zemeckis",
 "description": "Going back to the future"
 }
],
}`

4. You can use the visual indicators that are shown in the vertical ruler to expand or collapse a corresponding region in the editor. For example, you can expand or collapse structural elements of objects and arrays that span more than one line. To format large amounts of content, click an indicator to toggle the expanded or collapsed state of the corresponding region in the editor.
5. To collapse all regions, right-click the vertical ruler and select **Folding > Collapse All**. All regions are hidden in the editor.
6. To expand all regions, right-click the vertical ruler and select **Folding > Expand All**. All regions are shown in the editor.
7. Save your changes.

Setting JSON editor preferences

You can customize the display of the JSON editor by setting preferences such as highlight and syntax colors.

Procedure

1. Click **Window > Preferences > Web > JSON > Editor**.
2. On the Editor preferences page, you can enable or disable folding and the highlighting of brackets, and select the highlight color.
3. In the preferences navigation, expand **Editor** and select **Syntax Coloring**.
4. On the Syntax Coloring preference page, select the color and formatting for various JSON elements.
5. You can set more preferences in the Text Editors preference page. On the Editor preferences page, click the **Text Editors** link.
6. Useful options for working with JSON files:
 - Show line numbers
 - Highlight current line
 - Show range indicator
7. Click **Apply** and then **OK** to save your changes.

Creating a Dojo profiled build (custom build)

A Dojo build profile is a JavaScript file that specifies build parameters.

Before you begin

[Create a Dojo enabled Web project.](#)

Procedure

1. Click **File** > **New** > **Other** > **JavaScript** > **JavaScript Source File** and then click **Next**. The New JavaScript File wizard opens.
2. Select a parent folder from the folders list.
3. Type the name of your profile file with the `.profile.js` extension. **Tip:** The builder wizard automatically appends `.profile.js` to the name of the profile that you specify in the Dojo Build Tools wizard. This naming convention is mandatory by the wizard.
4. Click **Finish**. A new JavaScript file is created and opens in the editor.
5. Script your builder profile with your layers and prefixes dependencies. The contents of the file will look like:

```
dependencies:
{
  layers: [
    // Individual layer objects
  ]
  prefixes: [
    // Individual prefix objects
  ]
}
```

- dependencies

- (*Optional*) Contains layer and prefix members.

- layers

- (*Optional*) Array of layer objects that the build generates. Layers are built in the order in which they are listed. By convention the layers member precedes the prefixes member.

- prefixes

- (*Optional*) Array of prefix objects that describes the location of the source of a specific top-level module, relative to the source directory.

Note: If the profile does not contain any dependencies objects (it is empty), all defaults are applied to the build. The build creates a single layer file that contains only Dojo Base. All other resources must be loaded using `dojo.require`.

Example

Here is an example of the contents of a build profile file:

```
dependencies = {
  stripConsole: "normal",

  layers: [
    {
      name: "../dijit/dijit.js",
      dependencies: [
        "dijit.dijit"
      ]
    },
  ],
}
```

```

{
  name: "../dijit/dijit-all.js",
  layerDependencies: [
    "../dijit/dijit.js"
  ],
  dependencies: [
    "dijit.dijit-all"
  ]
},
{
  name: "../dojox/grid/DataGrid.js",
  dependencies: [
    "dojox.grid.DataGrid"
  ]
},
{
  name: "../dojox/gfx.js",
  dependencies: [
    "dojox.gfx"
  ]
},
{
  name: "../dojox/charting/widget/Chart2D.js",
  dependencies: [
    "dojox.charting.widget.Chart2D",
    "dojox.charting.widget.Sparkline",
    "dojox.charting.widget.Legend"
  ]
},
{
  name: "../dojox/dtl.js",
  dependencies: [
    "dojox.dtl",
    "dojox.dtl.Context",
    "dojox.dtl.tag.logic",
    "dojox.dtl.tag.loop",
    "dojox.dtl.tag.date",
    "dojox.dtl.tag.loader",
    "dojox.dtl.tag.misc",
    "dojox.dtl.ext-dojox.NodeList"
  ]
}
],

prefixes: [
  [ "dijit", "../dijit" ],
  [ "dojox", "../dojox" ]
]

```

}

What to do next

Now, you can [run a custom Dojo build](#).

Running a Dojo custom build

The Dojo build system creates an efficient version of Dojo for application deployment by creating Dojo builds that are customized to a specific web application.

Before you begin

1. [Create a Dojo-enabled web project.](#)
2. [Create a builder profile.](#)

About this task

The Dojo build system also improves performance:

- The Dojo build system assembles the Dojo resources and external dependencies, such as widget templates, into one or more layers. The contents of the external dependencies change into a string in the layer JavaScript file.
- The Dojo build system compresses the layers using ShrinkSafe to remove any extra spaces, extra lines, comments, duplicate resources, and to shorten internal variable names.
- The Dojo build system copies all non-layered JavaScript to an easily accessible location, ensuring that all Dojo resources can be loaded even if they are not contained by a layer.

Learn more about Dojo layers: The Dojo library contains many files and resources that can be called by the web page. When you use the `dojo.require` statement, the web page makes a synchronous HTTP call to the server to retrieve a resource. The `dojo.require` statement prevents the web page from having to load a resource that has been loaded prior; however, by using the `dojo.require` statement, your web page must first load the resource before it can be used. In a web page with multiple `dojo.require` statements, each call to the server must be completed before it moves to the next call, significantly affecting application performance.

A Dojo layer file reduces the number of asynchronous requests to a single request. A layer is a single JavaScript file that contains and compresses multiple JavaScript files and any dependencies. It can contain an entire library or it can contain all of the widgets that are requested by a particular page. All extra spaces, blank lines, and comments are removed and internal variable names are shortened. using the HTML `script` tag. The layer files are loaded asynchronously reducing the load time.

For example, you might have a web application with multiple Dojo require statements in each page. By creating a layer file for each page, only one asynchronous request is made per page.

For more information on the Dojo build system, refer to [Dojo build system documentation](#).

To enhance the performance of your Dojo application with the Dojo build system:

Procedure

1. Click **File > New > Other > Web > Dojo Custom Build** and then click **Next**. The Dojo Build Utility wizard opens.
2. Specify your **Profile Locations** file that you created in [Create a builder profile](#).
3. Specify the location of the Dojo library on the **Dojo Location** field.
4. Specify the build scripts and output directories, on the **Output Location** field. **Tip:** The profile, build scripts, and output directories can be in the same project or different projects.
5. To display only the layer files in the output, check **Only output layer files**.
6. Click **Override profile settings with command line** to specify an **Optimization** method. You can specify whether to delete output directories before building, copy test files into the build, or intern widget templates. When you intern a template, the HTML or CSS file is brought into the JavaScript file and assigned to a string.

7. (Optional) Click **Next** to specify advanced options.
 - To add a command-line argument, click the add icon. Type the argument and value in the New Argument window.
 - To edit an existing command-line argument, click the edit icon.
 - Click the delete icon to remove a command-line argument.
8. Click **Finish**. The Custom Build Output window opens and displays details of the build operation. Any error messages appear in red text.
9. Click **OK** to close the Custom Build Output window.

Results

The entire Dojo distribution is built and the Dojo layer files that you selected are created in the output folder that you specified in the wizard.

What to do next

Now that you created Dojo layers using the Dojo build system, you can [include a Dojo layer file in your web page](#).

Including a Dojo layer file in your web page

A layer is a single JavaScript file that contains and compresses an entire Dojo library and any dependencies. To reduce the amount of time it takes to retrieve a resource, you can load Dojo layers by using the HTML `script` tag. The layer files are loaded asynchronously reducing the load time.

Before you begin

1. [Run a custom Dojo build.](#)
2. [Create a Web page.](#)

About this task

To include a Dojo layer file in your web page, drag the layer file from Enterprise Explorer onto your web page.

Results

The layer is added to your file source. This is how it looks: `<script language="JavaScript" src="dojo.js"></script>`

What to do next

Now, you can [Add Dojo widgets to your web pages.](#)

Developing mobile web applications

The mobile web application development tools help you to create interactive web pages that are optimized for mobile devices.

About this task

For information on practices for developing mobile web applications, refer to [Mobile Web Application Best Practices](#).

For information about Dojo mobile widgets, see the mobile tools section of the [Dojo Toolkit API documentation](#). For demonstrations of the Dojo mobile widgets, visit the [Dojo Mobile Showcase](#).

Procedure

1. [Create the web project for your mobile application](#).
2. [Create the web pages for your mobile application](#).
3. [Add Dojo mobile widgets to your mobile web pages](#).
4. [Test your mobile web application](#).
5. [Publish your application to a server](#).

- [Adding Dojo mobile widgets to your mobile web page](#)

- [Editing web pages for mobile devices](#)

Parent topic: [Developing web applications](#)

Related information:

[Mobile Navigation view](#)

[Mobile patterns](#)

Adding Dojo mobile widgets to your mobile web page

Before you begin

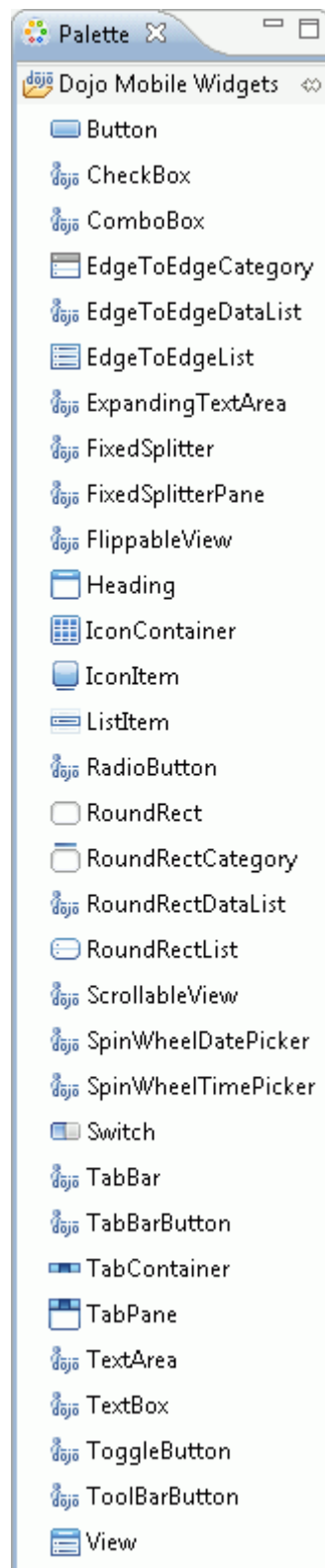
1. [Create a mobile web project.](#)
2. [Create web pages for mobile devices.](#)

About this task

For information about Dojo mobile widgets, see the mobile tools section of the [Dojo Toolkit API documentation](#). For demonstrations of the Dojo mobile widgets, visit the [Dojo Mobile Showcase](#).

Procedure

1. Open the web page in the editor.
2. Drag a widget from the Dojo Mobile Widgets drawer of the Palette view to the page in the editor.



Parent topic: [Developing mobile web applications](#)

Editing web pages for mobile devices

Before you begin

1. [Create a mobile web project.](#)
2. [Create web pages for mobile devices.](#)

Procedure

1. Open the web page in the Rich Page Editor.
2. Edit the page by dragging widgets from the Palette view onto the editor or code your changes in the Source view of the editor.

Parent topic: [Developing mobile web applications](#)

Adding tag library support for web projects

This product supports HTML, JSP, and JSF tag libraries. You can also add custom or third-party tag libraries to your web projects.

- **Tag library support for web projects**

When you drag a tag from the Palette onto a web page in the editor, all the required JAR files and resources are copied into your web project.

- **Adding support for a component library**

A component library contains components that you can use in your web applications to create a user interface or a web page. A library definition can contain custom JSP tags or third-party Dojo tags. A library definition includes file resources and metadata that can be distributed across your team.

- **Adding support for custom tag libraries**

A custom tag library is a set of custom tags that start custom actions in a web page. Tag libraries reduce the task of embedding excessive amounts of code in web pages by moving the functionality that is provided by the tags into tag implementation classes. Use this product to create custom Dojo widgets and JSP tags.

- **HTML and JSTL tag libraries**

Parent topic: [Developing web applications](#)

Tag library support for web projects

When you drag a tag from the Palette onto a web page in the editor, all the required JAR files and resources are copied into your web project.

This product supports the following tag libraries:

- [HTML and JSP tag libraries](#)
- [Custom tag libraries](#)
- [Third party libraries](#)

Tag libraries offer the following benefits:

- Help separate presentation from implementation.
- Are easy to maintain and reuse.
- Simplify complex actions.
- Provide Java-coded functions without the task of coding in Java™.
- Can dynamically generate page content and implement a controlled flow.

Parent topic: [Adding tag library support for web projects](#)

Adding support for a component library

A component library contains components that you can use in your web applications to create a user interface or a web page. A library definition can contain custom JSP tags or third-party Dojo tags. A library definition includes file resources and metadata that can be distributed across your team.

Before you begin

Ensure that the library JAR file for which you want to add support exists in your file system, your workspace, or in a component library project in your workspace.

Procedure

1. [Create a library project](#) to hold all of your library definitions.
2. [Configure support for your library definition.](#)

What to do next

After you add support for a component library, the library tags display in a drawer in the Palette view and you can add the tags to your web pages.

- [Creating a library project](#)

A component library contains components that you can use in your Web applications to create a user interface or a Web page. A library definition can contain custom JSP tags. A library definition includes file resources and metadata that can be distributed across your team. The library project holds all of the library definitions that you can use in your workspace.

- [Creating a JSP Library Definition \(CLD\)](#)

You can add a CLD file to your Library Definitions project in the JSP Library Definition wizard.

- [Updating a library definition](#)

The update process finds new tags and attributes in a tag library and adds them to the library definition.

- [Adding translations to a library definition](#)

You can add translated strings to your library definitions.

Parent topic: [Adding tag library support for web projects](#)

Creating a library project

A component library contains components that you can use in your Web applications to create a user interface or a Web page. A library definition can contain custom JSP tags. A library definition includes file resources and metadata that can be distributed across your team. The library project holds all of the library definitions that you can use in your workspace.

Procedure

1. Click **File > New > Other > Web > Library Definitions Project**. Click **Next**. The Create Library Definitions wizard opens. A Definition project can hold multiple library definitions.
2. In the Library name field, enter the name of the library and then click **Finish**.

Results

A library project is created in your workspace.

What to do next

Now that you created a library project you can create library definitions for JSP tags:[JSP Library Definition \(CLD\)](#)

Parent topic:[Adding support for a component library](#)

Creating a JSP Library Definition (CLD)

You can add a CLD file to your Library Definitions project in the JSP Library Definition wizard.

Procedure

1. Right-click your Library Definitions project and select **New > Other > Web > JSP Library Definition**. Click **Next**.
2. In the **Library name** field, enter the name of the library.
3. In the **Select a jar file** section, specify the location of the JAR file that contains the library.
4. In the **Specify a Library Definitions project where this definition will be stored** section, specify where you want to store the library definition. Click **Finish**.

Results

The JSP Library Definition wizard creates a new metadata file for the library and opens the file in the JSP Library Definition editor. A drawer, containing the tags in the library, is added to the Palette view so that you can drag a tag from the Palette view onto a web page.

What to do next

After you add support for the component library, you can [configure the library definition](#) in the editor.

- [Configuring the JSP Library Definition](#)

A JSP Library Definition contains project resources that are needed for a component library and the metadata necessary for the interpretation of JSP tags. It helps you to customize the integration of a custom JSP tag library into this product.

Parent topic: [Adding support for a component library](#)

Configuring the JSP Library Definition

A JSP Library Definition contains project resources that are needed for a component library and the metadata necessary for the interpretation of JSP tags. It helps you to customize the integration of a custom JSP tag library into this product.

Before you begin

[Add support for an existing component library.](#)

Procedure

1. Double-click your JSP Library Definition to open the library definition file in the editor. Your library definition file has a CLD extension.
2. Make any necessary changes to your library definition. The library definition file includes the following sections:
 - [JSP Library Definition](#)
 - [Application Configuration](#)
 - [Tag Library](#)
3. After you configure your library definition, save the file. The definition is updated.
4. To update the file, click **Update Library Definition** then click **Perform update**. The update process finds new tags and attributes in the tag library and adds them to the library definition. **Remember:** Increment the library definition version after the update is complete.

Parent topic: [Creating a JSP Library Definition \(CLD\)](#)

JSP Library Definition

The JSP Library Definition editor is used to configure the details about how to incorporate a new JSP library into the project.

About this task

- Library name

- Name of the library.

- Description

- A description of the library.

- Taglib URI

- The taglib URI of the component library.

- Prefix

- The prefix that is given to the tags.

- Version

- The version number indicates the level of the configuration in the library definition. It signals when newer resources might be available to existing web applications. When a library definition is initially created, pick a version that is meaningful for the selected tag library. For example, start with the release number for publicly available tag libraries. The typical form of the version number is a dot-separated string such as 1.2.3.4. After the library definition is in use and the resources for a tag library are installed in one or more web applications, further changes to the definition might need a version update. If any changes are made in the **Application Configuration** section of a deployed library definition, increase the version number. The increased version number indicates to the workbench that newer resources or configuration changes are available. Any web projects that use the tag library are marked with an error. The Quick Fix associated with that error can update the web projects with the current configuration. **Remember:** If a tag library is not added to a web project, increasing the version number is unnecessary.

Changes to any areas of the Library Definition other than the Application Configuration section are dynamically read and therefore do not require a version change.

Application Configuration

The following details how to set up the web project to use the new component library. The configuration declares how to copy files into the web project and any configuration changes that must be made to make components function properly.

About this task

- Resources

- The resources area is used to define files to copy into a web project when a library is first used. The source path is the relative path to a file within the JSP Definitions project. The target path is the path within the project where the library is used. For example, the file from source path `/mystylesheet.css` might be given a target of `/theme`. This target path results in a copy of the CSS file that is placed into the `/theme` folder of the web project.

- web.xml Updates

- There are several types of modifications that are made to the deployment descriptor of a web project where the library is used. New servlets, context parameters, and filters can be configured under corresponding subsections in the editor.
- To add a servlet, context parameter, or filter:
 1. Select **web.xml**.
 2. Click **Add** and select the item from the dialog box.

- Servlet

- Defines parameters for a new servlet to be added to web.xml.

- Context parameter

- Records the details for adding a context parameter in the web deployment descriptor. When the library is first used in a web project, the `web.xml` file is modified to include this information.

- Filter

- Defines a new filter, which is added into the `web.xml` file. Initialization parameters for this filter are added by clicking **Add** in the editor.

- Initialization parameter

- Collects data for an initialization parameter for a `web.xml` filter definition.

- URL Mappings

- Collects any URL-based mappings for the filter.

- Servlet Mappings

- Collects filter mappings that are based on a servlet name, which is defined in `web.xml`.

Tag Library

The Tag library lists all of the custom tags that make up the library and enable the configuration of tool behavior. For example, you can specify the appearance of tags in the Palette, behaviors when tags drop onto the editor, the visual appearance of components in web page source, and various details about tag attributes and data binding.

About this task

You can configure details about the tags in the component library. When the library definition is generated, it contains all of the tags in the component library. You can change the order of tags with the **Up** and **Down** buttons. The specified tag order is shown in the appearance of the library in the Palette.

Some visual information about the library palette category can be configured. Configure any style sheet links or JavaScript references that apply to all tags in a library here. When a tag is added from the palette, the corresponding `<link>` and `<script>` tags are added to the web page.

- Tag

- Configures the appearance of the component tag in the palette, including icon and label information. The tag can also be hidden by default or removed from the palette choices.

- Drop properties

- The behavior of inserting new tags from the palette is modified by parameters on this page. If the component is a container-like control and other components are allowed as children, select **Allows children**. Style sheet links, JavaScript references can be specified on a tag-by-tag basis on this page. Style sheet and JavaScript specifications can also be made for the entire library on the Tag page.

- Default tag attributes

- To set a default attribute value of a tag, specify the name and value of the attribute. When a component tag is added from the palette this attribute value is automatically set.

- Visualization

- For many components, the display of the tag in the visual page editor renders correctly. In some cases, the tag rendering fails entirely or it does not render appropriately. If the rendering fails, the tag appears as a gray box with the tag name. When this problem occurs, you can customize the appearance of the tag by specifying basic HTML tags with some variable modifiers. This tag is used as a substitute visualization in the page editor.
- Sample visualization templates are available and include examples of how to use the variable modifiers. To view the templates, click **Edit** in the Details section. **Learn more about visualization pattern variables:** A visualization pattern is used for controlling the appearance of a component in the design pane of the page editor. The pattern consists of HTML markup and some optional substitution variables to dynamically alter the content of the visualization. When a component is dragged onto a page, the corresponding pattern is evaluated. The page editor uses the resulting HTML to create a useful design-time representation. Details of the available pattern variables are shown in Table 1.

Table 1. Visualization pattern variables. The description and examples provide information about variables.

Variable	Description	Example
<code>\${children}</code>	The <code>\${children}</code> variable emits all the child tags of the tag being visualized, if any are present, at the position of the <code>\${children}</code> variable in the pattern. Child tags are inserted in the order they appear in the original page. The page editor computes the visualizations for the child tags since they can also use patterns. Since all child tags are inserted in the same location, a pattern can use only one <code>\${children}</code> variable.	<pre><div> \${children} </div></pre>

<code>#{children:row}</code>	<p>The <code>row</code> modifier of the <code>#{children}</code> variable emits the children tags in a sequence of HTML table cell <code><td></code> elements. Use this variable when you are arranging multiple components horizontally.</p>	<pre><table> <tr> #{children:row} </tr> </table></pre>
<code>#{children:grid(param[,param]*)}</code>	<p>The <code>grid</code> modifier creates a set of HTML table rows and cells that contain children of a tag. Children are added left-to-right and wrap to the next row when a specified number of columns is reached. The default column count is 2. The optional comma-separated parameter list overrides the number of columns in the grid.</p> <p>The parameter options are: <code>attr.attrName</code> The value of the tag attribute with the name <code>attrName</code>. This parameter must resolve to an integer. <code>N</code> A positive integer constant. Parameters are examined sequentially until a valid value is found. For example, the pattern</p> <pre>#{children:grid(attr.size, attr.numcols, 3)}</pre> <p>would use the <code>size</code> attribute of the tag if it was defined otherwise it would use the <code>numcols</code> attribute, if defined. If neither attribute is defined, the constant 3 is used.</p>	<pre><table> #{children:grid(attr.columns , 2)} </table></pre>

<pre> <code> $\{children:stack(parameter)\}$</code> </pre>	<p>The <code>stack</code> modifier is used in tabbed panels to create table rows and cells for populating a stacked-cell visualization. It creates a horizontal table of labels that can be clicked for each child tag. Clicking one of the labels brings the visualization of the child tag to the top of the stack. The optional parameter specifies which tag attribute of the child tags to use for the label, using the <code>attr.attrName</code> syntax.</p>	<pre> <code> <table> $\{children:stack(attr.label)\}$ </table> </code> </pre>
<pre> <code> $\{attr:attrName\}$ </code> </pre>	<p>This variable substitutes the attribute value of a tag at the current location in the pattern. If the attribute does not have a set value, an empty string is inserted.</p>	<pre> <code> <input type="text" value="$\{attr:value\}$" /> </code> </pre>

- Tag attributes

- For each tag, an attribute description and type can be specified. The order of the attributes in the list and the type for each affects how they are shown and edited in the Properties view. Certain types indicate a set of available choices or a specific helper dialog box to assist in setting an attribute value.

- Properties view

- This optional section contains user interface elements and layout information for a user-defined Properties View page. Nested child elements can be added beneath the Properties View section to define the contents of one or more tabs.

- Tab

- A tab contains other UI elements within it and corresponds to the Properties View. The tab names are shown in the view, with the main (first) tab initially displayed. The first tab always has the name of the selected tag so the value of the name attribute for the first tab is ignored.

- Column

- The Column element visually groups a set of other UI elements into a vertical column. Two or more columns are defined inside either a Tab or Section element and child controls are configured within each column. Non-column elements are not supported inside the same container where columns are used. You might be required to separate a multi-column arrangement into its own region within sections.

- Editor

- Editor elements define:
 - Which type of user interface controls are displayed on the Properties View, such as a radio button.

- Which attribute of the current tag the user interface control modifies.
- Which string label to show next to the user interface control.

You can also add parameters to the editor element.

- **Editor Parameter**

- Some editor elements can take optional editor parameters to customize the behavior of the control. For example, you can configure a combination box and other similar selection controls using the **choices** parameter that defines the allowable values. The value of the choices parameter uses the syntax `enum{value1, value2, value3}` or if separate labels and values are used in the combination box: `enum{name1:value1, name2:value2}`. Most of the other predefined editor parameters accept `true` or `false` values.

- **Label**

- The Label element places a label containing text information in the Properties View.

- **Section**

- The Section element is a container that defines a self-contained portion of the UI. Sections can be used to provide control over spacing of the controls they contain. For example, a section can be configured to span multiple columns in a surrounding layout, or to define a new column structure for its own contents.

- **Separator**

- The Separator element uses a horizontal rule to divide the Properties View contents.

Updating a library definition

The update process finds new tags and attributes in a tag library and adds them to the library definition.

Before you begin

[Add support for a component library.](#)

About this task

You can update the following component library definitions:

Procedure

1. Open the library definition file in the editor.
2. In the Enterprise Explorer, right-click your library and select **New > File**. Leave the default parent folder and add the file name of the JAR file you want to add.
3. Click **Advanced** and then **Link to file** in the file system if you need to.
4. Click **Finish**.
5. In the editor, expand the **Library Definition** in the Overview section. Click on **Resources** and then click **Add**.
6. Click on **File** and then **OK**.
7. In the Details section of the editor, in the source path field, click the **...** button and locate the source of the new JAR file to include.
8. Click **Update Library Definition** and then click **Perform update**. The JSP component library is updated.
9. Increment the library definition version number in the **Version** field.

Parent topic: [Adding support for a component library](#)

Adding translations to a library definition

You can add translated strings to your library definitions.

Before you begin

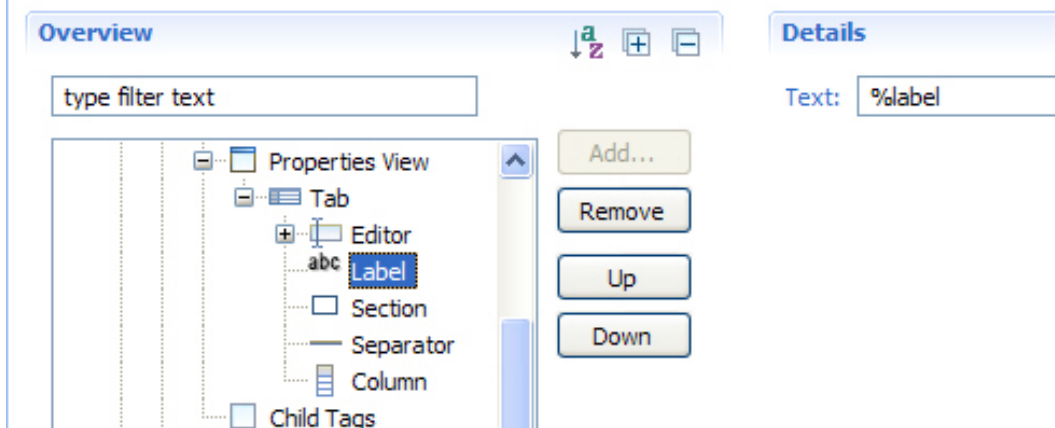
[Configure a JSP library definition.](#)

About this task

You can substitute translated string values from a *.properties file for certain values in a library definition. For example, you may want to have certain fields that are translated in the Properties view of a library definition, like Name or Label.

Procedure

1. Externalize the strings in the library definition that you want to have translated. In this example, the text attribute for the label element is externalized as %label.



2. Create a corresponding *.properties file and list the variables that you externalized. For example:

```
# NLS_MESSAGEFORMAT_VAR
```

```
# NLS_ENCODING=UTF-8
# =====
# Translation Instruction: section to be translated
# =====
label = label description
```

3. Place the English and the translated *.properties files into the same directory as the library definition file (*.jld or *.cld).

Note: The translated *.properties files must be in ASCII format. You may need to convert them by running the native2ascii Java™ utility on the translated *.properties files. The native2ascii utility converts a file with native-encoded characters (characters, which are non-Latin 1 and non-Unicode) to one with Unicode-encoded characters. For example, the properties file listed in step 2 might look like this when translated into another language:

```
# NLS_MESSAGEFORMAT_VAR
# NLS_ENCODING=UTF-8
# =====
# Translation Instruction: section to be translated
# =====
label = \uc11c\u3ec\u2c0\ubcf8\u81b \uc608\uc81c \ubb38
```

4. Rename the translated *.properties files with the appropriate locales if they are not already renamed. For example, filename_de.properties for a file that is translated into German, or filename_ja.properties for a file that is translated into Japanese.
5. To view the results of the new *.properties files, close the library definition file and restart the product.

Parent topic: [Adding support for a component library](#)

Adding support for custom tag libraries

A custom tag library is a set of custom tags that start custom actions in a web page. Tag libraries reduce the task of embedding excessive amounts of code in web pages by moving the functionality that is provided by the tags into tag implementation classes. Use this product to create custom Dojo widgets and JSP tags.

- **Creating custom Dojo widgets**

You can create custom Dojo widgets in the New Dojo Widget wizard.

- **Adding custom JSP tags**

You can create custom JSP tags for your web projects that are based on the Oracle JSP 1.2 Specification. Tag libraries enable you to enhance your website regardless of your proficiency in the Java™ programming language.

Parent topic: [Adding tag library support for web projects](#)

Adding custom JSP tags

You can create custom JSP tags for your web projects that are based on the Oracle JSP 1.2 Specification. Tag libraries enable you to enhance your website regardless of your proficiency in the Java™ programming language.

About this task

Implementing custom tags requires all of the following:

- Defining custom tags in a Tag Library Descriptor (TLD) file. This is analogous to defining XML tags in a DTD file. The TLD file is an XML file that describes the custom tags in a tag library and includes tag information, such as the tag names, type of content, attributes, and associated tag handler class.
- Using custom tags within a JSP page. To use a custom tag within a JSP page, you must first identify where the TLD file is located and identify a prefix to be used when any of the custom tags in the library are included in a JSP page. This is accomplished by using a taglib directive.
- Creating a Tag Handler class. This is a Java class that implements the Tag or BodyTag interface and is responsible for the implementation of a custom tag at runtime.

A TLD file can be packaged within a JAR file, or as a separately existing project file. If the TLD file is packaged in a JAR file, it must be included under the META-INF folder.

You can add your custom tag library and tag library descriptor files to your project as you would any other project file.

To add custom JSP tags, you typically follow this process:

Procedure

1. [Add the TLD file.](#)
2. [Add a taglib directive to the JSP file.](#)
3. [Specify the taglib directive.](#)
4. Optional: [Edit the custom tag properties.](#)
5. Optional: [Edit the web deployment descriptor file for the custom tag library.](#)

- Custom tag libraries

A custom tag library is a set of custom tags that invoke custom actions in a JavaServer Pages (JSP) file. Tag libraries move the functionality provided by the tags into tag implementation classes, reducing the task of embedding excessive amounts of Java code in JSP pages.

- Adding the Tag Library Descriptor (TLD) file

In order to implement custom JSP tags, you must add the Tag Library Descriptor (TLD) file (taglib.tld) to your project.

- Adding a taglib directive to a JSP file

Before you can insert custom tags in your JSP file, you must insert a JSP taglib directive into the JSP page to obtain a tag library location.

- Specifying the taglib directive

You need to specify the taglib directive in order to pick up the tags that you are going to put into the document.

- Editing the properties of a custom tag

After custom tags are inserted into your web pages, you can edit attribute names and values, and modify tag content in the Source page. If the content type is JSP, you can also edit in the Design or Split pane.

- Editing a web deployment descriptor file for a custom tag library

The web deployment descriptor file can contain a <taglib-uri> and a <taglib-location> tag that maps the URI to the actual TLD file. The <taglib-uri> serves as an alias for the TLD file that is published as WEB-INF/taglib.tld. This enables you to

shorten the taglib directive in the JSP file.

Parent topic:[Adding support for custom tag libraries](#)

Custom tag libraries

A custom tag library is a set of custom tags that invoke custom actions in a JavaServer Pages (JSP) file. Tag libraries move the functionality provided by the tags into tag implementation classes, reducing the task of embedding excessive amounts of Java™ code in JSP pages.

Tag libraries are created by developers who are proficient in the Java programming language. These libraries can be used by web designers who might not know Java, but want to enhance their website by taking advantage of Java encoded tag libraries.

Tag libraries offer the following benefits:

- Help separate presentation from implementation.
- Are easy to maintain and reuse
- Simplify complex actions
- Provide Java coded functions without the task of coding in Java.
- Can dynamically generate page content and implement a controlled flow.

You can develop tag libraries yourself or you can download them from existing open source utility tag libraries. For example, you can download tag libraries from sites such as the [Jakarta Project](#), a division of the Apache Software Foundation. Jakarta also offers a tag libraries tutorial on that site.

This product includes the JavaServer Pages Standard Tag Library (JSTL), a library of custom tags that provide the core functionality common to many web applications. If you include the JSF and JSTL project facets in your project, you can select and insert a particular tag from the tag library when you edit the JSP file. (To insert a custom tag, right click in the JSP and select **JSP > Insert Custom Tags**).

You can add the JSTL library to a project when you create the project by selecting the JSTL project facet from the New Web Project wizard. **Note:** If you did not include JSTL when you first created the project, you can add it afterward by selecting **Project > Properties > Project Facets**.

The *web.xml* file provides the link between the directive used in the application and the actual JAR file containing the classes that execute the function.

Parent topic: [Adding custom JSP tags](#)

Adding the Tag Library Descriptor (TLD) file

In order to implement custom JSP tags, you must add the Tag Library Descriptor (TLD) file (taglib.tld) to your project.

Procedure

1. Add the independent tag library descriptor, taglib.tld to your WEB-INF folder:
 - A. In the Enterprise Explorer view of the Web perspective, select the **WebContent/WEB-INF** folder in your web project.
 - B. Right-click the **WEB-INF** folder and select **Import**.
 - C. Click **General > File System** and then click **Next**.
 - D. In the File System page of the Import wizard, click **Browse** to select the directory from which you want to import your independent tag library descriptor, taglib.tld.
 - E. Select the tld file from the tree viewer on the wizard.
 - F. Click **Finish** to add your independent tag library to your WEB-INF folder.
2. Add the custom tag library to the web project:
 - A. In the Enterprise Explorer view of the Web perspective, right-click the web project and select **Import**.
 - B. Click **File System > Next**.
 - C. In the File System page of the Import wizard, click **Browse** to select the directory from which you want to import your custom tag library.
 - D. Select the library from the navigation tree in the wizard
 - E. Click **Finish** to add your custom tag library to your web project.

What to do next

After you add the TLD file to your project, you can [add a taglib directive to your JSP file](#).

Parent topic: [Adding custom JSP tags](#)

Adding a taglib directive to a JSP file

Before you can insert custom tags in your JSP file, you must insert a JSP taglib directive into the JSP page to obtain a tag library location.

Before you begin

[Adding the Tag Library Descriptor \(TLD\) file.](#)

Procedure

1. Open the JSP file in the Web Page Editor. Right-click it and select **Open with > Web Page Editor**.
2. Open the palette. Click **Window > Show view > Palette**
3. Drag and drop a Directive -taglib from the JSP Tags in the palette.
4. Click the Directive -taglib object in the editor. Go to the Properties view and open the Attributes tab.
5. Edit the properties of the directive by adding a prefix, tagdir, and uri.
6. Save the page.

What to do next

Now that you have added a taglib directive to your JSP file, you can [specify the taglib directive](#).

Parent topic: [Adding custom JSP tags](#)

Specifying the taglib directive

You need to specify the taglib directive in order to pick up the tags that you are going to put into the document.

Before you begin

1. [Add the Tag Library Descriptor \(TLD\) file.](#)

About this task

You can specify taglib directives in one of four ways:

Procedure

1. Using the taglib-uri value in the web deployment descriptor, as follows: `<%@ taglib uri="/yeartags" prefix="year" %>`

```
<%@ taglib uri="http://www.mycorp/monthtags" prefix="month" %>
```

where both `/yeartags` and `http://www.mycorp/monthtags` are taglib-uri values that are defined in the web deployment descriptor.

2. Using the context-relative path that refers directly to the TLD or JAR file, as follows: `<%@ taglib uri="/tlds/datetags.tld"`

```
prefix="date" %>
```

where `/tlds/datetags.tld` is a context-relative URI to the TLD file.

3. Using a page-relative path that refers directly to the TLD or JAR file, as follows: `<%@ taglib uri="../WEB-`

```
INF/tlds/hourtags.jar" prefix="hour" %>
```

where `../WEB-INF/tlds/hourtags.jar` is a page-relative URI to the JAR file.

4. For a Java™ EE 1.3 web project only, using the URI element value that is defined in the TLD, as follows: `<%@ taglib`

```
uri="http://www.mycorp/minutetags" prefix="minute" %>
```

where `http://www.mycorp/minutetags` is the URI element value that is defined in the TLD. **Tip:** In a case where two or more TLDs are deployed inside a JAR file, you can use this format to specify each TLD.

What to do next

Now that you have specified your taglib directive, you can add a custom tag to your JSP file. When you choose to insert a custom tag, Page Designer references the taglib directive and displays the tags from that custom library.

Parent topic: [Adding custom JSP tags](#)

Editing the properties of a custom tag

After custom tags are inserted into your web pages, you can edit attribute names and values, and modify tag content in the Source page. If the content type is JSP, you can also edit in the Design or Split pane.

Before you begin

1. [Add the Tag Library Descriptor \(TLD\) file.](#)
2. [Add a taglib directive to a JSP file.](#)
3. [Specify the taglib directive.](#)

Procedure

1. To edit attributes of a custom tag, use the Source pane.
2. To modify the content of a custom tag whose content is tagdependent, use the Source pane.
3. To modify the content of a custom tag whose content is JSP, you can use both the Design and Source panes.

Results

By default, custom tags are display as the **Custom tag** icon  in the Design pane. You can customize the appearance of the icon using preferences. Select **Windows > Preferences > Web > Page Design > Appearance > Editing Symbols**.

You can select one of the following options to determine how custom tags are displayed in the Design pane: JSP:

- **Show with icon**

- **Show with icon and text**

Note: In JSP 1.2 TLD files, each custom tag can have an optional small-icon that can be used by tools. If a small icon is specified for a custom tag, the icon is used in the Design pane in place of the default icon.

Example

This is a portion of a JSP file that displays a custom tag that has been modified. The Name attribute has been edited.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0.1 Transitional//EN">
<HEAD>
<META name="GENERATOR" content="IBM Page Designer">
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE>Hello World!</TITLE>
<%@ taglib uri="mytags" prefix="mt" %>
</HEAD>
<HR>
<mt:helloWorld name="foo"/>
<HR>
```

Parent topic: [Adding custom JSP tags](#)

Editing a web deployment descriptor file for a custom tag library

The web deployment descriptor file can contain a `<taglib-uri>` and a `<taglib-location>` tag that maps the URI to the actual TLD file. The `<taglib-uri>` serves as an alias for the TLD file that is published as `WEB-INF/taglib.tld`. This enables you to shorten the taglib directive in the JSP file.

Before you begin

1. [Add the Tag Library Descriptor \(TLD\) file.](#)
2. [Add a taglib directive to a JSP file.](#)

Procedure

1. Expand your web project in the Enterprise Explorer view of the web perspective.
2. Right-click Deployment Descriptor and select **Open**. The web deployment descriptor editor opens.
3. Click the tabs in the editor to edit the deployment descriptor. For more information about these tabs, refer to topics on a [web deployment descriptor editor](#). Use the Source page to edit the web deployment descriptor source directly.

Example

This is an example of `<taglib-uri>` and `<taglib-location>` tags used in a web deployment descriptor file:`<?xml version="1.0"`

```
encoding="UTF-8"?>
```

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
```

```
</web-app>
```

```
<display-name>helloWorld-server</display-name>
```

```
<taglib>
```

```
    <taglib-uri>mytags</taglib-uri>
```

```
    <taglib-location>WEB-INF/taglib.tld</taglib-location>
```

```
</taglib>
```

```
</web-app>
```

Parent topic: [Adding custom JSP tags](#)

HTML and JSTL tag libraries

HyperText Markup Language (HTML)

HTML is a language for publishing hypertext on the web. This product supports all of the tags in the HTML 4.01 Specification. For more information about HTML tags, see [HTML 4.01 Specification](#).

In addition to the HTML 4.01 Specification, this product also supports some non-standard tags. For more information about these non-standard tags, see [Non-standard HTML tags](#).

JavaServer Pages Standard Tag Library (JSTL)

This product incorporates a custom tag library (JavaServer Pages Standard Tag Library or JSTL) from the Jakarta project. Many convenient tags are provided in the JSTL, which includes several tag libraries:

- Core tags: Flow control, for example loops and conditional statements, and general-purpose actions.
- XML tags: Enable basic XML processing within a JSP.
- Formatting tags: Internationalized data formatting.
- SQL tags: Database access for querying and updating.
- Function tags: Various string handling functions.

You can incorporate JSTL as a project facet when you create a web project, or you can add it to the project afterward by selecting **Project Properties Project Facets**. For more information about the JSTL tag library, including the API specification, see [JavaServer Pages Standard Tag Library](#).

After you create the project, a web deployment descriptor shortcut is generated and follows your project name. This shortcut serves as an alias for the `web.xml` file, and defines the library tag names and attributes available to your project. The tag library JAR files are placed in the `WEB-INF/lib` folder.

The Libraries folder contains a representation of the tag libraries available to your application. Empty JAR icons indicate tag libraries that are internal to your application (the tag libraries that you selected in the New Project wizard). Full JAR icons indicate tag libraries that are external to your application (the tag libraries that are not copied to your workspace).

Non-standard HTML tags

The following is a list of non-standard HTML tags. To specify particular actions, you can add [Standard attributes](#) that apply to all tags or you can apply attributes that are particular to individual tags. Click each tag for descriptions and other attributes that apply to the specific tags:

- `<bgsound>`
- `<blink>`
- `<embed>`
- `<marquee>`
- `<nobr>`
- `<noembed>`
- `<wbr>`

<bgsound>

`<bgsound>` specifies a sound file for the page.

Attribute	Attribute options	Description	Editing and display options
-----------	-------------------	-------------	-----------------------------

loop	infinite, number_of_repeats	Specifies the number of times that a sound file is repeated. Specify <i>infinite</i> to play the sound file infinitely or a number.	You can edit this attribute on the Source page or in the Outline view. Cannot display on the Design page.
src		Specifies the file name or URL of the sound file that you want to use in your document.	You can edit this attribute on the Source page or in the Outline view. Cannot display on the Design page.

<blink>

<blink> specifies text that flashes on and off. You can edit and display this tag in the Design view; however, you cannot view the flashing text on the Design page. This tag is indicated by the dotted lines that surround the contents.

<embed>

<embed> embeds a plug-in object in the document. This tag can be edited and displayed on the Design page.

Attribute	Description	Editing and display options
align	Specifies the alignment of the object.	Can be edited using the Format menu, the Insert menu, or in the Attributes view. Can be displayed on the Design page.
flashvars	Specifies the variable to pass to a Flash player. Requires Macromedia Flash Player 6 or later. It is used to send root level variables to the movie. The format of the string is a set of name=value combinations separated by &. Browsers will support string sizes of up to 64KB (65535 bytes) in length.	You can edit this attribute in the Embed Flash Plugin dialog when you drag the Flash component onto a web page. You can also edit this attribute in the Source.
height	Specifies the height of the object. The units are specified in the units attribute.	Can be edited using the Format menu, the Insert menu, or in the Attributes view. Can be displayed on the Design page.

palette	Specifies the background or foreground color.	You can edit this attribute on the Source page or in the Outline view.
pluginspage	Identifies the location of the Flash player plug-in so that a user can download the player if it is not already installed.	You can edit this attribute in the Embed Flash Plugin dialog when you drag the Flash component onto a web page. You can also edit this attribute in the Source.
quality	Specifies the quality of the playback. Values include: low Favors playback speed over appearance and never uses anti-aliasing. medium Applies some anti-aliasing and does not smooth bitmaps. It produces a better quality than the low setting but a lower quality than the high setting. The default value. high Favors appearance over playback speed and always applies anti-aliasing. If the movie does not contain animation, bitmaps are smoothed. If the movie has animation the bitmaps are not smoothed. auto low Emphasizes speed during the beginning of playback but improves appearance whenever possible. Playback begins with anti-aliasing turned off but can be turned on when the processor can handle the additional load. auto high Emphasizes playback speed and appearance equally at first but then sacrifices appearance for playback speed if necessary. Playback begins with anti-aliasing turned on. If the actual frame rate drops below the specified frame rate, anti-aliasing is turned off to improve playback speed. best The best display in terms of speed and appearance.	You can edit this attribute in the Embed Flash Plugin dialog when you drag the Flash component onto a web page. You can also edit this attribute in the Source.

src	Specifies the URL of the object to be embedded.	Can be edited using the Format menu, the Insert menu, or in the Attributes view.
text	Specifies a title for the object.	You can edit this attribute on the Source page or in the Outline view.
type	Specifies the data type.	You can edit this attribute on the Source page or in the Outline view.
width	Specifies the width of the object. The units are specified in the units attribute.	Can be edited using the Format menu, the Insert menu, or in the Attributes view. Can be displayed on the Design page.

<marquee>

<marquee> defines text that moves in a document. This tag can be edited and displayed on the Design page. It runs only on Internet Explorer 4.0 or later.

Attribute	Attribute options	Description	Editing and display options
behavior	scroll, slide, alternate	Specifies the manner in which the text moves.	Can be edited using the Format menu or the Insert menu.
bgcolor		Specifies the background color of the marquee.	Can be edited using the Format menu or the Insert menu. Can be viewed on the Design page.
direction	left, right, down, up	Specifies the direction in which the text moves.	Can be edited using the Format menu or the Insert menu.
height		Specifies the height of the marquee in pixels.	Can be edited using the Format menu or the Insert menu. Can be displayed on the Design page.
loop		Specifies whether the marquee loops.	Can be edited using the Format menu or the Insert menu.
scrolldelay		Specifies the scroll delay speed.	Can be edited using the Format menu or the Insert menu.
scrollamount		Specifies the scroll speed.	Can be edited using the Format menu or the Insert menu.
truespeed		Displays the marquee at the specified speed.	Can be edited using the Format menu or the Insert menu.

width		Specifies the width of the marquee in pixels.	Can be edited using the Format menu or the Insert menu. Can be displayed on the Design page.
hspace		Specifies the horizontal margins in pixels.	Can be edited on the Source page or in the Outline view. Cannot be displayed on the Design page.
vspace		Specifies the vertical margins in pixels.	Can be edited on the Source page or in the Outline view. Cannot be displayed on the Design page.

<nobr>

<nobr> overrides line breaks in a block of text. This tag can be edited on the Source page or in the Outline view, but cannot be displayed on the Design page.

<noembed>

<noembed> specifies alternate content for browsers that do not support the plug-in required for an inline media type. This tag can be edited on the Source page or in the Outline view, but cannot be displayed on the Design page.

<wbr>

- Purpose

- Enables a line break within a <NOBR> tag. You can edit this tag on the Source page or in the Outline view, but you cannot display it on the Design page.

Standard attributes

You can apply Standard attributes to all HTML tags to specify more information about a tag or to specify particular actions. Standard attributes fall into four categories: Core attributes, Keyboard attributes, Language attributes, and Event attributes.

Core attributes enable you to provide context information about a tag, for example, you could include text to describe a tag.

Attribute	Description
id	Assigns a specific name for a tag in a document.
class	Specifies one or more classes to which a tag belongs.
style	Applies style information to a tag.
title	Defines text that describes the contents of a tag or its contents.

Keyboard attributes enable you to specify options that related to accessing the HTML elements with the keyboard.

Attribute	Description
accesskey	Defines a keyboard shortcut to access an element in your document.
tabindex	Defines the order in which an element is tabbed in your document.

Language attributes enable you to specify options related to the language used by the text within tags, for example, you could specify the direction in which the characters appear.

Attribute	Attribute options	Description
lang		Indicates the language being used in the document.
dir	lrt, rtl	Indicates the direction of the text in the document.

Event attributes specify action when a user clicks on an HTML element.

Attribute	Description
onclick	An event occurs when you click the mouse on an element.
ondblclick	An event occurs when you double-click the mouse on an element.
onmousedown	An event occurs when you hold the mouse button over an element.
onmouseup	An event occurs when you release the mouse button over an element.
onmouseover	An event occurs when you drag your mouse over an element.
onmousemove	An event occurs when you move the mouse that is already over an element.
onmouseout	An event occurs when you move the mouse from an element.
onkeydown	An event occurs when you press and release a key over an element.
onkeypress	An event occurs when you press a key over an element.
onkeyup	An event occurs when you release a key over an element.

Parent topic: [Adding tag library support for web projects](#)

Debugging web applications

You can debug your web applications on a server to detect and diagnose errors in your application.

Procedure

- [Debug a servlet on a server.](#)
- [Debug a JSP file on a server.](#)
- [Debug web pages with Firebug.](#)

- **Debugging web pages with Firebug**

Firebug is an open source extension for Mozilla Firefox that assists you with debugging, editing, profiling, logging, and monitoring HTML pages, DOM, CSS, and JavaScript.

Parent topic: [Developing web applications](#)

Debugging web pages with Firebug

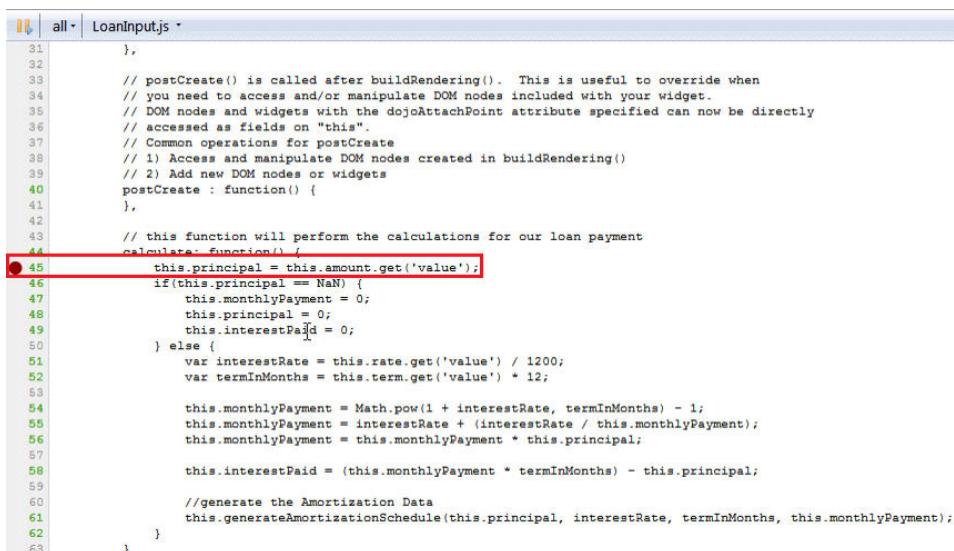
Firebug is an open source extension for Mozilla Firefox that assists you with debugging, editing, profiling, logging, and monitoring HTML pages, DOM, CSS, and JavaScript.

Before you begin

1. [Create a web project.](#)
2. [Create a web page.](#)
3. [Install Mozilla Firefox.](#)
4. [Install Firebug.](#)

Procedure

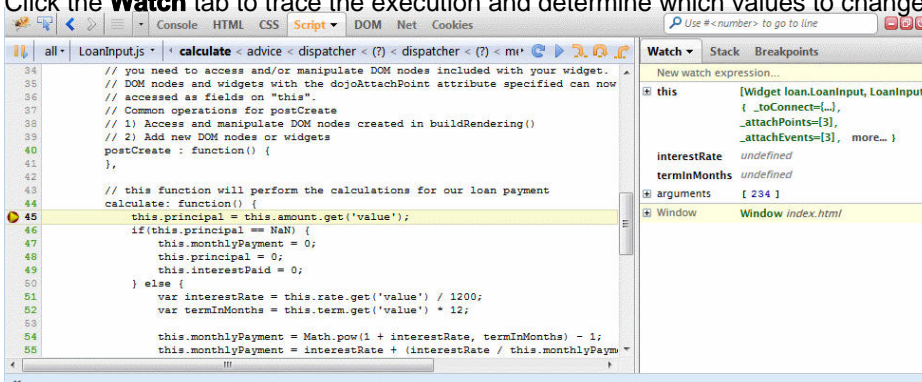
1. Enable Firebug by selecting **Window > Web Browser > Firefox** as your default browser in your workspace.
2. In the Enterprise Explorer view, right-click the web page that you want to debug and select **Debug As > Debug on Server**.
3. Click **Finish**. Your web page opens in Firefox.
4. To enable Firebug, click .
5. In the Firebug section, click the **Script** tab and then click **Enable**.
6. After the scripts are enabled, click **Reload**. The web application is ready for debugging.
7. Continue to use the debugging tools to step through the code.
8. In the JavaScript resource that you want to debug, set a breakpoint.



```
31     },
32
33     // postCreate() is called after buildRendering(). This is useful to override when
34     // you need to access and/or manipulate DOM nodes included with your widget.
35     // DOM nodes and widgets with the dojoAttachPoint attribute specified can now be directly
36     // accessed as fields on "this".
37     // Common operations for postCreate
38     // 1) Access and manipulate DOM nodes created in buildRendering()
39     // 2) Add new DOM nodes or widgets
40     postCreate : function() {
41     },
42
43     // this function will perform the calculations for our loan payment
44     calculate: function() {
45         this.principal = this.amount.get('value');
46         if(this.principal == NaN) {
47             this.monthlyPayment = 0;
48             this.principal = 0;
49             this.interestPaid = 0;
50         } else {
51             var interestRate = this.rate.get('value') / 1200;
52             var termInMonths = this.term.get('value') * 12;
53
54             this.monthlyPayment = Math.pow(1 + interestRate, termInMonths) - 1;
55             this.monthlyPayment = interestRate + (interestRate / this.monthlyPayment);
56             this.monthlyPayment = this.monthlyPayment * this.principal;
57
58             this.interestPaid = (this.monthlyPayment * termInMonths) - this.principal;
59
60             //generate the Amortisation Data
61             this.generateAmortizationSchedule(this.principal, interestRate, termInMonths, this.monthlyPayment);
62         }
63     },
64 }
```

The breakpoint that is set stops the execution according to the logic of your web page.

9. Click the **Watch** tab to trace the execution and determine which values to change.



Variable	Value
this	[Widget loan.LoanInput, LoanInput] { _toConnect=[...], _attachPoints=[3], _attachEvents=[3], more... }
interestRate	undefined
termInMonths	undefined
arguments	[234]
Window	Window index.html

Tip: You can enable, disable, or remove breakpoints from the **Breakpoints** tab.

- **Setting Firebug preferences**

Parent topic: [Debugging web applications](#)

Setting Firebug preferences Procedure

1. Click **Window > Preferences > General > Web Browser**.
2. Click **Use external Web browser** then select **Firefox** from the External Web browsers list.

What to do next

You can [debug your web pages](#).

Parent topic:[Debugging web pages with Firebug](#)

Configuring web applications

You configure web applications through the deployment descriptor or through annotations.

About this task

Web 2.2, 2.3, 2.4, or 2.5 web projects have a `web.xml` file in the project `WEB-INF` directory. Configure the `web.xml` file to specify deployment information for modules in the web development environment.

Web 3.0 and later web projects do not require a `web.xml` file. You can configure annotations to specify deployment information. Optionally, you can generate a deployment descriptor stub if a Web 3.0 web project requires a web deployment descriptor.

Procedure

- For a Web 2.2, 2.3, 2.4, or 2.5 web project, configure the deployment descriptor. See [Configuring web applications using the Web deployment descriptor editor](#).

If a Web 3.0 web project requires a web deployment descriptor, right-click your project and select **Java EE > Generate Deployment Descriptor Stub**.

- For a Web 3.0 or later web project, configure annotations. See [Adding annotations to a web application](#).

- [Adding annotations to a web application](#)

Java EE 5 and 6 support the injection of annotations into your source code, so that you can embed resources, dependencies, services, and lifecycle notifications in your source code, without having to maintain these artifacts elsewhere. Annotations simplify the development and configuration of enterprise applications.

Parent topic: [Developing web applications](#)

Configuring web applications with the web deployment descriptor editor

You can configure the deployment descriptor for web projects that have Web 2.5 or Web 3.0 modules in a web deployment descriptor editor. You can also use the editor to generate a deployment descriptor stub for a Web 3.0 module. For Web 2.4 and earlier modules, use a source XML editor.

About this task

When you create a Web 2.2, 2.3, 2.4, or 2.5 web project, the New web Project wizard places a `web.xml` file in the project `WEB-INF` directory. This file specifies deployment information for modules that are created in the web development environment.

The editor that you use to specify deployment information in the `web.xml` file depends on the specification version of the web module.

Procedure

- For Web 2.2, 2.3 or 2.4 modules, use a source XML editor to edit `web.xml` file for these modules.
- For Web 2.5 or Web 3.0 modules, use a web deployment descriptor editor to construct the deployment descriptor for your web application rather than editing the source. Though the web deployment descriptor editor provides a source page of the `web.xml` file (from the **Source** tab), using the editor is easier and introduces fewer errors. As you work within the editor, the `web.xml` file is updated automatically.
 1. Open the `web.xml` file in a web deployment descriptor editor. Double-click the `web.xml` file or double-click the **Deployment Descriptor** icon in the Enterprise Explorer view.
 2. Configure the web deployment descriptor for your web project. See the topic for a Web 3.0 or Web 2.5 module:
 - [Web 3.0 Modules: The Web Deployment Descriptor Editor](#)
 - [Web 2.5 Modules: The Web Deployment Descriptor Editor](#)
- If a Web 3.0 web project requires a web deployment descriptor, generate a deployment descriptor stub. Web 3.0 web projects do not require a `web.xml` file because deployment information is managed using annotations. However, you can generate a deployment descriptor stub. Right-click your project and select **Java EE > Generate Deployment Descriptor Stub**.

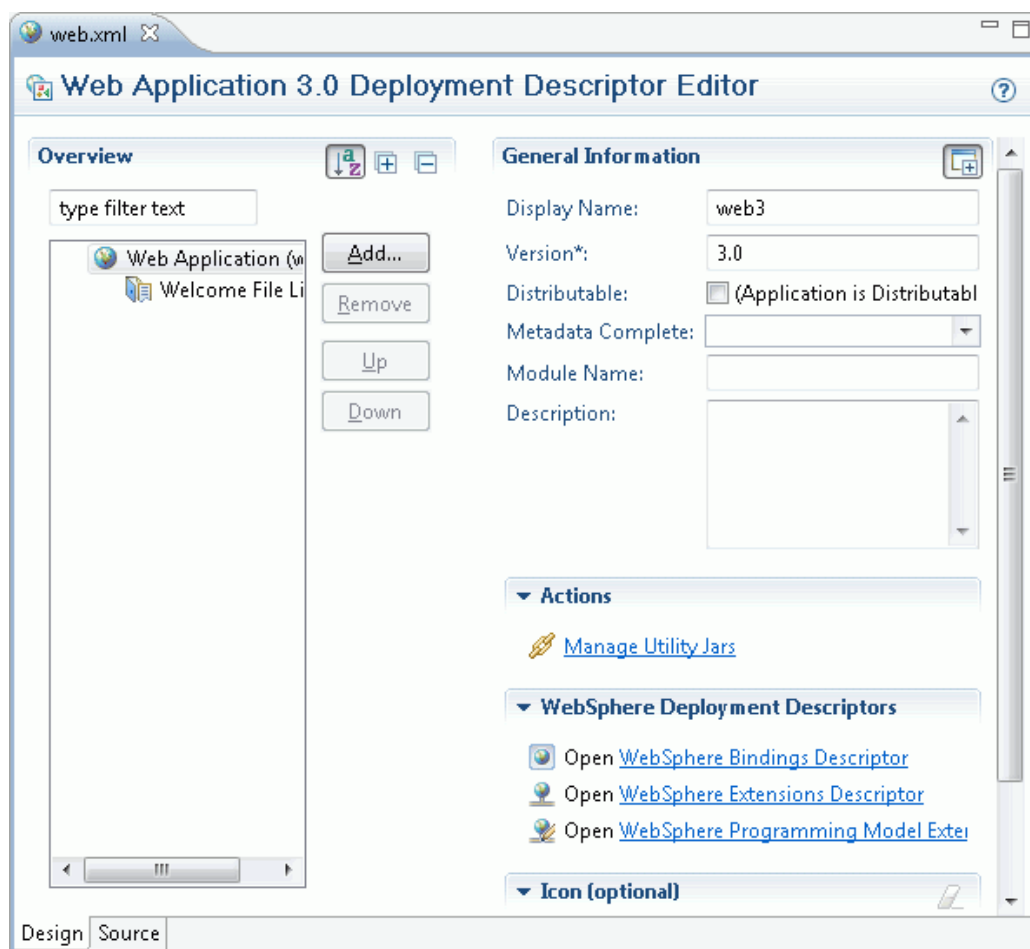
Web 3.0 Modules: The web Deployment Descriptor editor

Web 3.0 web projects do not require a `web.xml` file since deployment information is managed by annotations. If a web deployment descriptor is required for your web 3.0 web project, right-click your project and select **Java EE > Generate Deployment Descriptor Stub**.

The web deployment descriptor editor helps you specify deployment information for modules that are created in the web development environment. The information appears in the `web.xml` file. **Note:** You use the web deployment descriptor to set deployment descriptor attributes. You do not use it to manipulate web resource content.

The `web.xml` file for a web project provides information necessary for deploying a web application module. It is used in building a WAR file from a project.

The web deployment descriptor editor is dynamic and includes many sections that represent various properties and settings in the deployment descriptor.



The web deployment descriptor editor includes the following sections:

- Web Application Structure

- Provides a quick summary of the contents in the web deployment descriptor and allows you add, remove, or change the contents.

- Absolute Ordering

- Use absolute ordering to force an order for scanning web fragments or scan a subset of the web fragments.

- Context parameters

- A servlet context defines a server's view of the web application within which the servlet is running. The context also enables a servlet to access resources available to it.
 - Using the context, a servlet can log events, obtain URL references to resources, and set and store attributes that other servlets in the context can use. These properties declare a web application's parameters for its context. They convey setup information, such as a webmaster's email address or the name of a system that holds critical data.
 - **Data source**
 - Configure data source information to make the data source available to the application.
 - **EJB Local Reference**
 - Create a reference to an enterprise bean that is accessed through its local home and local interface.
 - **EJB Reference**
 - Create a reference to an enterprise bean that is accessed through its remote home and remote interface.
 - **Environment Variable**
 - Declare an environment entry for the application.
 - **Error Page**
 - Error page locations enable a servlet to find and serve a URI to a client based on a specified error status code or exception type.
 - These properties are used if the error handler is another servlet or JSP file. The properties specify a mapping between an error code or exception type and the path of a resource in the web application. The container examines the list in the order that it is defined, and attempts to match the error condition by status code or by exception class. On the first successful match of the error condition, the container serves back the resource that is defined in the Location property.
 - **Filter**
 - Defines a filter class and its initialization attributes.
 - Creates a new filter, adds an existing filter to the deployment descriptor, or removes the selected filter from the deployment descriptor.
 - **Filter Mapping**
 - Defines the filter mapping to a URL pattern or servlet.
 - **JSP Configuration**
 - Add a resource collection or a tag library.
 - **Listener**
 - Defines an application listener.
 - **Locale Encoding Mapping List**
 - Maps locale name to an encoding name.
 - **Login Configuration**
 - Configures how a user is authenticated. If **Login Configuration** is specified, the user must authenticate in order to access resources that are constrained by the **Security Constraint** parameter.
 - **Message Destination**
 - Specifies the destination of a message-driven bean.
 - **Message Destination Reference**
 - Specifies the Java™ Naming and Directory Interface (JNDI) name of the J2C administered object to bind the message destination reference to the message-driven beans. Map each message destination reference in your application to an administered object.
 - **MIME Mapping**
 - Defines the mapping between an extension and a mime type.
 - **Persistence Context Reference**
 - Specifies the lifetime of the persistence context.
 - **Persistence Unit Reference**
-

- Specifies the `persistence.xml` file.
- Post Construct**
 - Defines the method that is run after dependency injection initialization.
- Pre Destroy**
 - Defines the callback notification that signals that the instance is being removed by the container.
- Resource Environment Reference**
 - Defines the reference of a resource in the web application to an associated an administered object.
- Resource Reference**
 - Defines the reference of a lookup name to an external resource.
- Security Constraint**
 - Security constraints determine how web content is to be protected. These properties associate security constraints with one or more web resource collections.
 - A constraint consists of a web resource collection, an authorization constraint, and a user data constraint.
 - A web resource collection is a set of resources (URL patterns) and HTTP methods on those resources. All requests that contain a request path that matches the URL pattern that is described in the web resource collection are subject to the constraint. If no HTTP methods are specified, then the security constraint applies to all HTTP methods.
 - An authorization constraint is a set of roles that users must be granted in order to access the resources that are described by the web resource collection. If a user who requests access to a specified URI is not granted at least one of the roles that are specified in the authorization constraint, the user is denied access to that resource.
 - A user data constraint indicates that the transport layer of the client or server communications process must satisfy the requirement of either guaranteeing content integrity (preventing tampering in transit) or guaranteeing confidentiality (preventing reading while in transit).
- Security Role**
 - Defines security roles.
- Security Reference**
 - Defines a reference to a security role to an alternate role name.
- Servlet**
 - Creates a new servlet, adds an existing servlet or JSP file to the deployment descriptor, or removes the selected servlet or JSP file from the deployment descriptor.
- Session Configuration**
 - Defines session attributes for the web application.
- Welcome File List**
 - Configures an ordered list of default web pages that are served when a file is not found.
 - A Welcome file is an entry point file (for example, `index.html`) for a group of related HTML files.
 - Welcome files are located by using a group of partial URIs. The web container uses the partial URIs to find a valid file when the initial URI is not found.
- Source**
 - Edit the `web.xml` source directly.

Edit the `web.xml` file by using the multiple tabbed pages in the web Deployment Descriptor editor. As you specify deployment information, the editor automatically incorporates the appropriate XML tagging in `web.xml`.

In addition to the configuration information in the `web.xml` file, other deployment descriptors in a web project include the following information:

- Binding information**
 - Information is required by the application server to bind the deployment information that is specified in the application to a specific instance. For example, it can map a logical name of an external dependency or resource to the actual physical JNDI name of the resource. It also can map security role information to a set of groups or users.

- IBM® binding and extensions information (`ibm-web-bnd.xml` and `ibm-web-ext.xml` files)

- Additions to the standard descriptors for Java EE applications, web applications, and enterprise beans. The extensions enable Java Platform, Enterprise Edition or legacy (older) systems to work in the current WebSphere® Application Server environment. They are also used to specify application behavior that is vendor-specific, undefined in a current specification, or expected to be included in a future specification.

If you import a WAR file into an existing web project, you can include the deployment descriptor files that are included in the WAR file as the web project's new deployment descriptor. Any specific deployment information that is already defined in these files is used when deploying the updated web application.

The `web.xml` file can be updated automatically to reflect changes to your web project. For instance, when you use the New servlet wizard to create a new servlet in a web project, the wizard places the appropriate servlet entry into the `web.xml` file.

Generating web deployment descriptors for web 3.0 projects

Web 3.0 web projects do not require a `web.xml` file since deployment information is managed by annotations. If a web deployment descriptor is required for your web 3.0 web project, you can generate one using the Generate Deployment Descriptor Stub.

Procedure

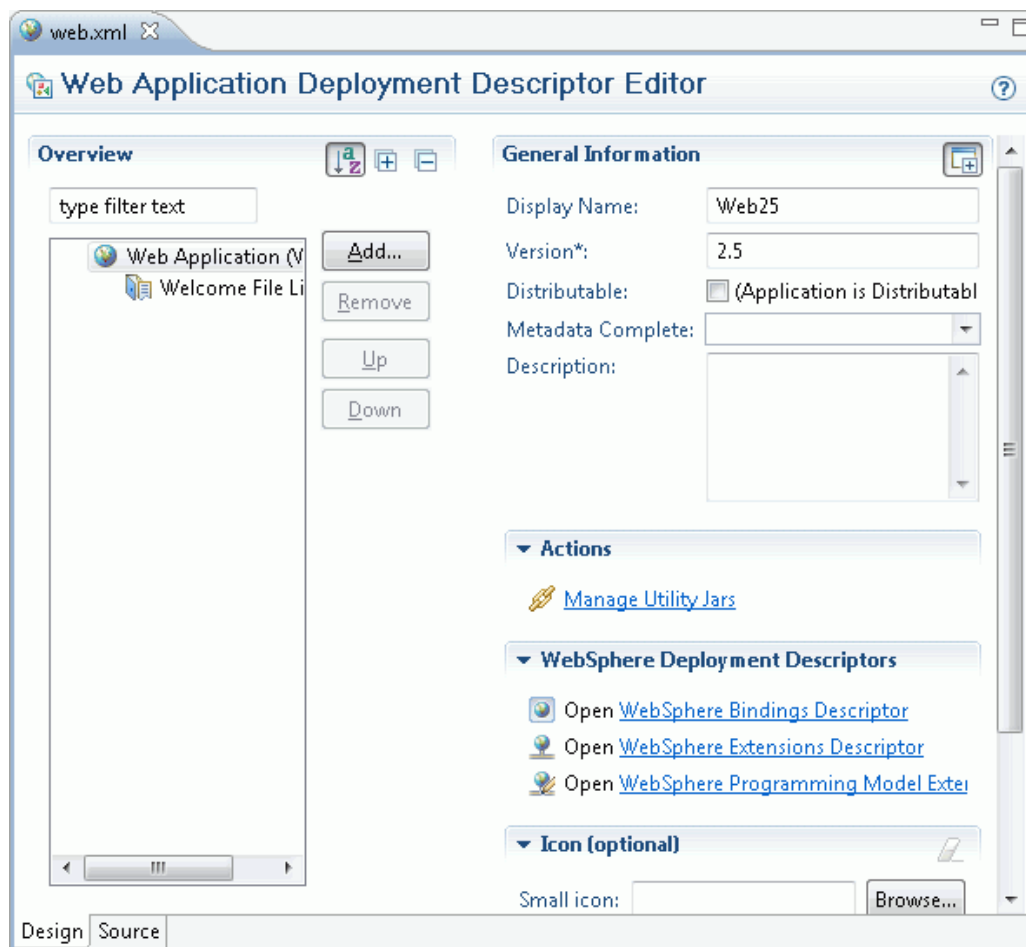
1. Right-click your web 3.0 web project and select **Java EE Tools > Generate Deployment Descriptor Stub**. The `web.xml` file is generated in your project `WEB-INF` directory.
2. Double-click `web.xml` to open the deployment descriptor in the editor.

Web 2.5 Modules: The web Deployment Descriptor editor

The web deployment descriptor editor helps you specify deployment information for modules that are created in the web development environment. The information appears in the `web.xml` file. **Note:** You use the web deployment descriptor to set deployment descriptor attributes. You do not use it to manipulate web resource content.

The `web.xml` file for a web project provides information necessary for deploying a web application module. It is used in building a WAR file from a project. Whenever you create a new web project, a minimal `web.xml` file is automatically created in `WEB-INF` under the project's web content folder.

The web deployment descriptor editor is dynamic and includes many sections that represent various properties and settings in the deployment descriptor.



The web deployment descriptor editor includes the following sections:

- **Web Application Structure**
 - Provides a quick summary of the contents in the web deployment descriptor and allows you add, remove, or change the contents.
- **Context parameters**
 - A servlet context defines a server's view of the web application within which the servlet is running. The context also enables a servlet to access resources available to it.

- Using the context, a servlet can log events, obtain URL references to resources, and set and store attributes that other servlets in the context can use. These properties declare a web application's parameters for its context. They convey setup information, such as a webmaster's email address or the name of a system that holds critical data.

EJB Local Reference

- Create a reference to an enterprise bean that is accessed through its local home and local interface.

EJB Reference

- Create a reference to an enterprise bean that is accessed through its remote home and remote interface.

Environment Variable

- Declare an environment entry for the application.

Error Page

- Error page locations enable a servlet to find and serve a URI to a client based on a specified error status code or exception type.
- These properties are used if the error handler is another servlet or JSP file. The properties specify a mapping between an error code or exception type and the path of a resource in the web application. The container examines the list in the order that it is defined, and attempts to match the error condition by status code or by exception class. On the first successful match of the error condition, the container serves back the resource that is defined in the Location property.

Filter

- Defines a filter class and its initialization attributes.
- Creates a new filter, adds an existing filter to the deployment descriptor, or removes the selected filter from the deployment descriptor.

Filter Mapping

- Defines the filter mapping to a URL pattern or servlet.

JSP Configuration

- Add a resource collection or a tag library.

Listener

- Defines an application listener.

Locale Encoding Mapping List

- Maps locale name to an encoding name.

Login Configuration

- Configures how a user is authenticated. If **Login Configuration** is specified, the user must authenticate in order to access resources that are constrained by the **Security Constraint** parameter.

Message Destination

- Specifies the destination of a message-driven bean.

Message Destination Reference

- Specifies the Java™ Naming and Directory Interface (JNDI) name of the J2C administered object to bind the message destination reference to the message-driven beans. Map each message destination reference in your application to an administered object.

MIME Mapping

- Defines the mapping between an extension and a mime type.

Persistence Context Reference

- Specifies the lifetime of the persistence context.

Persistence Unit Reference

- Specifies the `persistence.xml` file.

Post Construct

- Defines the method that is run after dependency injection initialization.

Pre Destroy

- Defines the callback notification that signals that the instance is being removed by the container.

Resource Environment Reference

- Defines the reference of a resource in the web application to an associated an administered object.

Resource Reference

- Defines the reference of a lookup name to an external resource.

Security Constraint

- Security constraints determine how web content is to be protected. These properties associate security constraints with one or more web resource collections.
- A constraint consists of a web resource collection, an authorization constraint, and a user data constraint.
- A web resource collection is a set of resources (URL patterns) and HTTP methods on those resources. All requests that contain a request path that matches the URL pattern that is described in the web resource collection are subject to the constraint. If no HTTP methods are specified, then the security constraint applies to all HTTP methods.
- An authorization constraint is a set of roles that users must be granted in order to access the resources that are described by the web resource collection. If a user who requests access to a specified URI is not granted at least one of the roles that are specified in the authorization constraint, the user is denied access to that resource.
- A user data constraint indicates that the transport layer of the client or server communications process must satisfy the requirement of either guaranteeing content integrity (preventing tampering in transit) or guaranteeing confidentiality (preventing reading while in transit).

Security Role

- Defines security roles.

Security Reference

- Defines a reference to a security role to an alternate role name.

Servlet

- Creates a new servlet, adds an existing servlet or JSP file to the deployment descriptor, or removes the selected servlet or JSP file from the deployment descriptor.

Welcome File List

- Configures an ordered list of default web pages that are served when a file is not found.
- A Welcome file is an entry point file (for example, `index.html`) for a group of related HTML files.
- Welcome files are located by using a group of partial URIs. The web container uses the partial URIs to find a valid file when the initial URI is not found.

Source

- Edit the `web.xml` source directly.

Edit the `web.xml` file by using the multiple tabbed pages in the web Deployment Descriptor editor. As you specify deployment information in these sections, the editor automatically incorporates the appropriate XML tagging in `web.xml`. In addition to the configuration information in the `web.xml` file, other deployment descriptors in a web project include the following information:

- Binding information

- Information is required by the application server to bind the deployment information that is specified in the application to a specific instance. For example, it may map a logical name of an external dependency or resource to the actual physical JNDI name of the resource. It also may map security role information to a set of groups or users.

- IBM® binding and extensions information (`ibm-web-bnd.xml` and `ibm-web-ext.xml` files)

- Additions to the standard descriptors for Java EE applications, web applications, and enterprise beans. The extensions enable Java Platform, Enterprise Edition or legacy (older) systems to work in the current WebSphere® Application Server environment. They are also used to specify application behavior that is vendor-specific, undefined in a current specification, or expected to be included in a future specification.

If you import a WAR file into an existing web project, you can include the deployment descriptor files that are included in the WAR file as the web project's new deployment descriptor. Any specific deployment information that is already defined in these files is used when you deploy the updated web application.

The `web.xml` file can be updated automatically to reflect changes to your web project. For instance, when you use the New Servlet wizard to create a new servlet in a web project, the wizard places the appropriate servlet entry into the `web.xml` file.

Adding annotations to a web application

Java EE 5 and 6 support the injection of annotations into your source code, so that you can embed resources, dependencies, services, and lifecycle notifications in your source code, without having to maintain these artifacts elsewhere. Annotations simplify the development and configuration of enterprise applications.

About this task

You can add annotations into your source code by using the Annotations view or by adding the annotation directly in the Java™ editor.

Procedure

1. Determine which annotations to add to your web application. *Table 1. Annotations supported by Java EE 5 web applications. This table describes common, EJB, and web service annotations for use in web applications.*

Annotation type	Annotation	Description
Common annotations	@Resource	The Resource annotation declares a reference to a resource that is required by the application.
	@Resources	The Resources annotation declares multiple resources declarations.
	@DeclaresRoles	The DeclaresRoles annotation specifies the security roles for the application.
	@RunAs	The RunAs annotation specifies the role of the application during run time.
	@PostConstruct	The PostConstruct annotation specifies the container that will start after resource injection is complete to perform any initialization.
	@PreDestroy	The PreDestroy annotation is used to signal that the instance is in being removed by the container.
EJBs	@PersistenceContext	The PersistenceContext annotation specifies the container managed entity context.
	@PersistenceContexts	The PersistenceContexts annotation declares multiple @PersistenceContext annotations.
	@PersistenceUnit	The PersistenceUnit annotation specifies a reference to an entity manager factory for use with EJBs.
	@PersistenceUnits	The PersistenceUnits annotation declares multiple @PersistenceUnit annotations.

Web services	@WebServiceRef	The WebServiceRef annotation specifies a reference to a web service within a web application.
	@WebServicesRefs	The WebServicesRefs annotation declares multiple @WebServiceRef annotations.

2. Add annotations in the source code by using the Annotations view or by adding the annotation directly in the Java editor.

- [Add annotations using the Annotations view.](#)

- [Add annotations manually.](#)

What to do next

For more information about web tier resource injection, refer to:

- [JSR 250: Common Annotation for the Java Platform Specification](#)

- [JSR 220: Enterprise JavaBeans 3.0 Specification](#)

- [JSR 224: Java API for XML-based Web Services \(JAX-WS\) 2.0 Specification](#)

Parent topic: [Configuring web applications](#)

Testing and publishing web applications

The testing and publishing tools provide runtime environments where you can test JSP files, servlets, HTML files, enterprise beans, and Java™ classes.

About this task

You can use workbench tools to publish and then test your web application.

Procedure

1. Use the workbench to publish your web application. The following topics describe how to publish using the workbench:

- [Testing applications on a server](#)
- [Publishing settings for WebSphere Application Server](#)

2. Test your web application. The following topics describe how to test web applications:

- [Testing mobile applications](#)

You can use the mobile browser simulator to emulate various mobile devices and test your mobile applications without the need to install the device vendor SDK.

- [Creating Dojo Object Harness \(DOH\) tests for Dojo test automation](#)

Dojo Object Harness (DOH) test tools help you evaluate your JavaScript and your web page user interface. Using the DOH test tools, you can create automated tests for your Dojo web applications.

- [Testing Dojo applications](#)

You can use the Dojo Object Harness (DOH) runner to test your Dojo application.

- [Testing on a Web Preview Server](#)

Use the Web Preview Server when you want to compile, test, and run resources quickly in an Ajax application. For example, you can test a Dojo-enabled web project on a Web Preview Server.

Parent topic: [Developing web applications](#)

Testing mobile applications

You can use the mobile browser simulator to emulate various mobile devices and test your mobile applications without the need to install the device vendor SDK.

Before you begin

1. [Create a mobile web project.](#)
2. [Add Dojo mobile widgets to your mobile web page.](#)

About this task

Important: The Mobile Browser Simulator supports the following web browsers:

- Firefox version 3.6 and later.
- Chrome 17 and later.
- Safari 5 and later

For more information about the mobile browser simulator, see [Mobile browser simulator](#).

Procedure

1. In Eclipse select **Window > Preferences > General > Web Browser > Use external web browser.**
2. Right-click your Application folder name and select **Run As > Run on Mobile Browser Simulator.**

What to do next

After your web page is running in the mobile browser simulator, you can view how your page renders in different devices.

-  [Switching devices](#)
-  [Adding devices](#)

- [Notes](#)

- [Mobile browser simulator](#)

The mobile browser simulator is a web application that helps you test mobile web applications without having to install device vendor native SDK.

- [Calibrating the mobile browser simulator](#)

Since browsers cannot accurately paint physical dimensions, you must calibrate the mobile browser simulator.

- [Enabling user agent switching](#)

You can use the mobile browser simulator to render your web applications on different mobile devices. To render your web applications with the appropriate style sheets and theme, you must enable user agent switching.

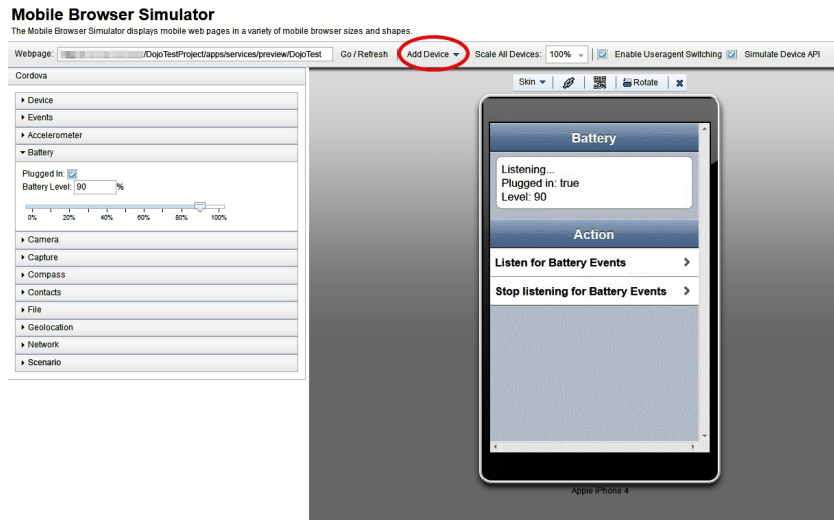
Parent topic: [Testing and publishing web applications](#)

Switching devices Before you begin

To view your web application in the simulated devices by using the appropriate style sheets, ensure that you [Enable user agent switching](#).

Procedure

In the simulator, click the device list and then select the device that you want to simulate.



Adding devices Before you begin

To view your web application in the simulated devices by using the appropriate style sheets, ensure that you [Enable user agent switching](#).

Procedure

In the simulator, click **Add Device** and then select the device that you want to simulate. **Tip:** You can customize the list of device options that are available in the mobile browser simulator.

1. Select **Windows > Preferences > Web > Target Devices**.
2. Add your custom device to the current list of target devices, and then start the simulator again.

Results

The custom device that you added is now available as an option from the **Add Device** list in the simulator.

Calibrating the mobile browser simulator

Since browsers cannot accurately paint physical dimensions, you must calibrate the mobile browser simulator.

Before you begin

[Test your application using the mobile browser simulator.](#)

About this task

For more information about the mobile browser simulator, refer to [Mobile browser simulator](#).

Procedure

1. From the **Scale All Devices** list, select **Physical device size**.
2. Click **Calibrate Physical Size** to open the Physical Size Calibration dialog.
3. Follow the instructions in the dialog to calibrate your mobile browser simulator. After you complete all of the steps in the dialog, close the dialog.

Parent topic: [Testing mobile applications](#)

Enabling user agent switching

You can use the mobile browser simulator to render your web applications on different mobile devices. To render your web applications with the appropriate style sheets and theme, you must enable user agent switching.

Before you begin

- [Test your application by using the mobile web simulator.](#)

About this task

The Useragent Switcher Extension is a browser extension that provides the user agent switching feature. The mobile browser simulator supports implementations of this browser extension for the following web browsers:

- Mozilla Firefox.

- Chrome 17 and later, with limitations.

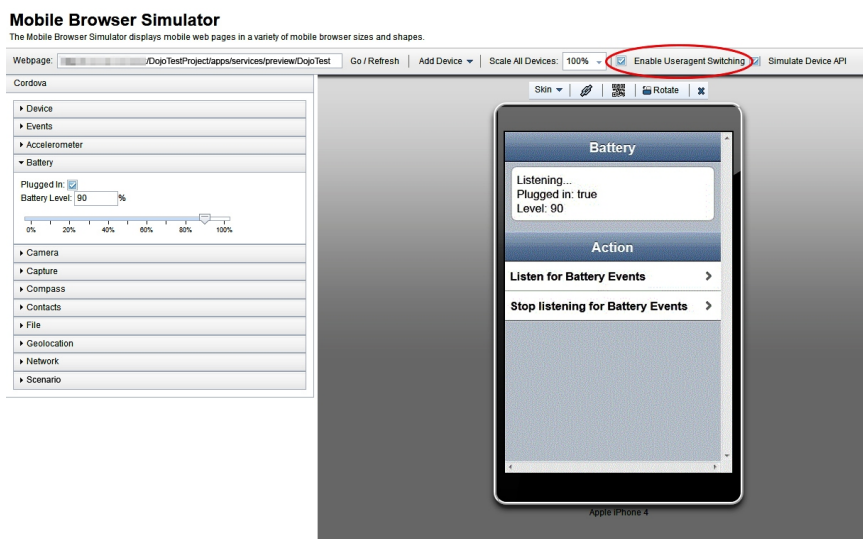
- Useragent Switcher Extension for Chrome

- The Useragent Switcher Extension emulates requests from different device-specific agents. When a web application checks the user agent on the server to create content, it is correctly simulated. The Useragent Switcher Extension includes support for Dojo Mobile 1.7 and later. If you enabled the detect device option when you created your Dojo Mobile page, the Useragent Switcher Extension uses the automatic device detection and theme loading for Dojo Mobile to select the appropriate theme.

For more information about the mobile browser simulator, see [Mobile browser simulator](#).

Procedure

1. Click **Enable Useragent Switching**.
2. If the latest version of the Useragent Switcher Extension is not installed, the Install Useragent Switcher Extension dialog opens. Click **Install Browser Extension**. **8.5.5.5** If you are using Chrome, you can download the extension from the Chrome Web Store.



Results

You can now view your web application with the appropriate style sheets and theme in the simulated mobile devices.

Parent topic: [Testing mobile applications](#)

Creating Dojo Object Harness (DOH) tests for Dojo test automation

Dojo Object Harness (DOH) test tools help you evaluate your JavaScript and your web page user interface. Using the DOH test tools, you can create automated tests for your Dojo web applications.

Before you begin

1. [Create a Dojo-enabled web project.](#)
2. [Create a Dojo widget.](#)

Procedure

1. In the Enterprise Explorer, right-click the Dojo class or widget that you want to test and then select **New > DOH test**. The New DOH test wizard opens.
2. In the Class under test, if your class name is not listed, type the initial letter and click **Browse** to select it. In the Available methods list, select the methods that you want to test. A test method is generated for each method that you select.
3. In the Types of test field, select the type of file that you want to use for the test:
 - **HTML - UI support (mostly used to test Dijit widgets)**
 - The generated test is an HTML file. This type of test is selected when testing Dojo widgets to include DOM elements in the test.
 - **JavaScript - No UI support**
 - The generated test is a JavaScript file. This type of test is selected when testing Dojo classes.
4. Click **Finish** to generate the DOH test. The Dojo module path mappings and the test file are generated. The test file contains one test method for each method that is selected in the wizard and contains asynchronous support (if the asynchronous support was selected). If the DOH module does not exist, the module is created. If the DOH module exists, an inclusion statement is added to the DOH module file to include the test as part of the test suite.
5. For each method that you selected in the Available methods list, add code to test the method.
6. Save the file.

What to do next

[Test your Dojo application.](#) For more information on the Dojo Object Harness, refer to the [Dojo Toolkit documentation for DOH](#).

- [Configuring DOH test properties](#)
- [Creating stand-alone DOH test launch](#)

You can create a DOH stand-alone test launch file to run DOH tests. The file redirects the browser to the URL of the DOH runner file and passes the required parameter for the test to the server.

Parent topic: [Testing and publishing web applications](#)

Configuring DOH test properties

Before you begin

[Create a Dojo-enabled web project.](#)

Procedure

1. Right-click the project for which you want to configure the DOH test properties and select **Properties**. The Properties dialog box opens.
2. In the properties list, click **DOH configuration**.
 - **DOH tests root folder**
 - Location of DOH tests for the project. The root folder must be located in `/WebContent`.
 - **DOH runner folder**
 - Location of the DOH runner file.
 - DOH tests run on the DOH runner platform. The DOH runner is an HTML file that contains logic to receive DOH tests as arguments to run them by providing an execution context. The DOH runner is not always included in all Dojo builds. For example, the Dojo build might be hosted at a CDN or the build is a compressed Dojo build without the test packages. When the Dojo build does not include the DOH runner, the **Get runner** button is available and you must select a runner file.**Important:** This option is disabled if you are using Dojo Toolkit SDK Version 1.9 or a later version.
 - **Dojo module path mappings**
 - Specifies the current module path mappings that are applied when DOH tests are run.
 - The Dojo module path mappings configuration is related to the Dojo mechanism for locating classes through the registration of module paths.
3. Click **OK** to save your changes.

Parent topic: [Creating Dojo Object Harness \(DOH\) tests for Dojo test automation](#)

Creating stand-alone DOH test launch

You can create a DOH stand-alone test launch file to run DOH tests. The file redirects the browser to the URL of the DOH runner file and passes the required parameter for the test to the server.

Before you begin

1. [Create a Dojo-enabled web project.](#)
2. [Create a Dojo widget.](#)

Procedure

1. Right-click a DOH module and select **New > New DOH runTests.html**. The New DOH runTests file wizard opens.
 - **DOH module class**
 - The DOH module class name that is run using the created file. It is read only.
 - **File name**
 - Name of the created file.
 - **Target folder**
 - Location of the file.
2. Edit the fields, if required, and then click **Finish**.

Parent topic: [Creating Dojo Object Harness \(DOH\) tests for Dojo test automation](#)

Testing Dojo applications

You can use the Dojo Object Harness (DOH) runner to test your Dojo application.

Before you begin

[Create a DOH test.](#)

Procedure

1. In the Enterprise Explorer view, right-click the DOH test and select **Run As > DOH Runner**. The Run On Server wizard opens.
2. Select the server that you want to use and then click **Finish**. The test runs and the results open in your web browser.

Parent topic: [Testing and publishing web applications](#)

Testing on a Web Preview Server

Use the Web Preview Server when you want to compile, test, and run resources quickly in an Ajax application. For example, you can test a Dojo-enabled web project on a Web Preview Server.

About this task

The Web Preview Server is a lightweight Liberty-based server ideally suited for developing and testing Ajax applications. It quickly performs module publishing and server restart. It supports basic web artifacts such as servlets, JSP files, HTML, XML, JavaScript, CSS files, and JAX-RS. It contains a proxy module that can be used to create Ajax proxy requests. External project references can be used by modules during both the development and runtime stages on this server. You can use the Web Preview Server as a convenient way to check for errors before you do a final test on an external application server. The server supports incremental publishing and hot deployment. You can change existing artifacts or add new artifacts to a running server without stopping and restarting the application server. If you change the Java™ source files in a referenced project, the server automatically restarts and loads the changes.

A Web Preview Server is automatically created for you when you open a new or existing workspace. If you want to create a Web Preview Server for previewing your resources in a web project, complete the following steps:

Procedure

1. In the Servers view, right-click and select **New > Server**.
2. Under the **Select the server type** list, expand the **IBM** folder and select **Web Preview Server**. Click **Next**.
3. In the Add and Remove page, under the Available projects list, select the web project and click **Add**. Click **Finish**. The Web Preview Server is created in the Servers view.
4. In the Project Explorer view, navigate to your web project and right-click your resource you want to test.
5. Select **Run As > Run on Server**.
6. In the Run on Server wizard, verify that **Choose an existing server** is selected.
7. Under the **Select the server that you want to use** list, select the recently created Web Preview Server. Click **Next**.
8. The **Add and Remove** page opens. Verify the web project that you are testing is listed under the **Configured** list. Click **Finish**.

Results

The internal web browser opens and shows the web content. By default, the Web Preview Server uses port 8080. If that port is already in use, a new port is automatically assigned. If you create multiple instances of the Web Preview Server, each server instance has its own copy of the Ajax proxy and unique port settings. The resources in the web project you are testing must be located under the directory that serves as the context root of the project.

- [Configuring a web Preview Server](#)

You can configure your web Preview Server and the `proxy-config.xml` file.

Parent topic: [Testing and publishing web applications](#)

Related tasks:

[Creating a server](#)

Configuring a web Preview Server

You can configure your web Preview Server and the `proxy-config.xml` file.

Procedure

1. In the Servers view, double-click the web Preview Server to open the server editor. You can use the Overview and Security pages to configure your web Preview Server.
2. In the General Information section of the Overview page, edit the server name, host name, and runtime environment.
3. In the Communication Settings section, specify the default port number to use for the web Preview Server. **Note:** If the default port number is already in use, the web Preview Server automatically assigns another available port number.
4. In the Publishing section, select one of the following settings:

Option	Description
Never publish automatically	The workbench never publishes files to the server.
Automatically publish when resources change	The workbench publishes when changes to a file that is associated with the server are saved and the full-time interval from the Publishing interval setting has passed. This is the default setting.
Automatically publish after a build event	The workbench publishes when changes to a file that requires a build and is associated with the server are saved, and the full-time interval from the Publishing interval setting has passed.
Publishing interval (in seconds)	The number of seconds before the workbench publishes the files on the server. However, if you make a subsequent change to the files before this time interval completes, publishing is delayed because the timer is reset. The workbench publishes to the server only after the full-time interval passes. If you set the publishing interval to 0 seconds, a file is published as soon as changes to a file are saved.

5. In the Timeouts section, set the amount of time for requests to complete.
6. The Ajax Proxy section contains a table that shows the access URLs with their corresponding resolved target URLs. The Ajax proxy section has a default proxy rule for wildcards that are listed in the **Access URL** column (`proxy/http/*`) which you can use to proxy any URL. For example, you can use this `http://localhost:8080/proxy/http/google.com` URL in the web browser to load the Google web page. This request that is mapped through the proxy has the same outcome as issuing a direct request to `http://www.google.com`. **Tip:** You can right-click on a row of the Ajax proxy URL table and copy the Access URL, for example `proxy/us/en` to paste into your JavaScript code.
7. Click the **Configure Proxy URLs** link to edit the `proxy-config.xml`.
8. In the Ajax Proxy Configuration editor, add proxy rules for mapping to remote domains. Select the **Proxy Rules** node and click **Add**.
9. In the Add Item window, select one of the following items and edit its attributes in the editor:

Option	Description
--------	-------------

Mapping	<p>Maps incoming requests to a target URL, based on their context path. Specify a Context path attribute and an optional URL attribute. For example, you might set a context path of <code>/ibmproducts</code> to the URL <code>http://www.ibm.com/products</code>. If the incoming proxy URL is <code>http://mywebsite.com/ibmproducts/us/en</code>, the proxy forwards the request to <code>http://www.ibm.com/products/us/en</code>. The proxy resolves context path mappings before you apply the matching access policy. Two default proxy mappings, <code>http/*</code> and <code>us/en</code>, are provided to support rapid prototyping.</p>
----------------	--

<p>Metadata</p>	<p>Specifies general configuration properties of the proxy, for example, HTTP-related parameters. The proxy editor provides some common metadata options with default name and value pairs:</p> <p>forward-http-errors By default, the Ajax proxy forwards only HTTP status codes greater than or equal to 200 and less than 400. Status codes that fall outside the range automatically change to a 404 File Not Found error. You can forward HTTP codes greater than or equal to 400 with a message by setting the forward-http-errors parameter to true.</p> <p>unsigned_ssl_certificate_support Unsigned certificates are often used on protected REST services. When unsigned_ssl_certificate_support is enabled, the Ajax proxy accepts any SSL certificate. This option is useful for testing and debugging. Do not use this option in a production environment.</p> <p>basic-auth-support Set this option to true if the target service uses basic authentication. An HTTP status code of 401 (unauthorized) results in 403: Forbidden HTTP code unless the basic-auth-support attribute is enabled for that specific request.</p> <p>maxconnectionsperhost The maxconnectionperhost is a global value that specifies the maximum number of connections kept alive for any host or port combination. By default, the value is set to 2. Increase the value if your application accesses more than two remote sites for content.</p> <p>maxtotalconnections The maxtotalconnections is the maximum total of connections that are supported by the proxy. The default value is 5. The value that you choose must be a high enough to support the number of simultaneous connections you might receive. In practice, factor in how the web container is configured and how many simultaneous connections the container supports.</p> <p>socket-timeout The socket-timeout defines the default socket timeout in milliseconds for waiting for data after a connection is established. The default is a timeout value of 0, which is interpreted as an infinite timeout.</p> <p>retries The retries parameter defines the number of socket retries the Ajax proxy performs before giving up on establishing a connection. The default value is 2.</p> <p>connection-</p>
------------------------	---

	<p>default value is 2. connection-timeoutThe connection-timeout defines the time in milliseconds before a connection is established. If no value is specified, then the default value of 60000 is used. If 0 is used, then the value is interpreted to mean that no timeout is used.</p>
<p>Policy</p>	<p>Defines an access policy for a given URL pattern. In the ACF field, turn active content filtering on or off. Active content filtering removes potentially malicious active content from application content that is displayed in a browser. Specify the pattern in the URL attribute field. For each incoming request, the proxy applies the policy with the best URL match. If no matching policy is found, the proxy rejects the request. If a policy is found, the subelements of the policy element are applied to check whether to accept the request. Turn support for basic authentication challenges on or off. You can edit the following policy subelements: Action methods Specify at least one supported HTTP method. The Ajax proxy supports GET, POST, PUT, HEAD, or DELETE requests. Cookies Define a list of cookie names that identify the cookies that you want the proxy to forward to the target domain. To forward cookies, the proxy filters the value of the cookie header according to the defined cookie names. HTTP headers Define the list of header names that you want the proxy to forward to the target domain. The header names can include wildcard characters. Mime-types Specify the list of accepted mime types. The mime types refer to the response that the proxy receives from the target server. If at least one mime type is specified, the proxy accepts only responses with a Content-Type response header that matches one of the specified mime types. If no mime type is specified, the proxy accepts all responses. Users Specify a list of users or user groups. For example, if you specify AllAuthenticatedUsers, the proxy verifies that the request was sent by an authenticated user.</p>

10. The Ajax Proxy keystore section shows information for the keystore that is used by the Ajax proxy, such as the keystore path and type. You can add or remove SSL certificates from the keystore to use the Ajax proxy to retrieve information from URLs for which the target server uses untrusted SSL certificates. After you import the SSL certificate of a URL that points to a server with an untrusted certificate, you can use Ajax Proxy to access this URL without receiving an error message.

- A. Optional: If the keystore password was changed by using software that is acquired from another vendor, update the password in the **Keystore password** field.
- B. Add or remove SSL server certificates from the keystore that is specified in the **Keystore path** field.
1. Click **Manage Keystore Entries**. The Manage Keystore window opens and shows a list of the current keystore entries, each identified by their alias.
 2. To remove a keystore entry, select the entry from the list of keystore entries, and then click **Remove Entry**.
 3. To add a keystore entry, click **Add Entry**. The Import Certificate window opens.
 4. In the **Entry Alias** field, specify an alias for the new keystore entry.
 5. Specify whether you want to import SSL certificates from a server or the local system.
 - **Import certificates from a server to the keystore**
 - Enter the URL of the server from which you want to import the SSL certificates.
 - **Import a local certificate to the keystore**
 - Click **Browse Certificate** to locate the SSL certificate on your local system that you want to import.
 6. Click **OK** to save your changes and close the Import Certificate window. In the Manage Keystore window, click **OK** save all of your changes to the keystore.
11. In the Security page, configure the following settings.

Section	Description
General	Enable or disable the security for web applications; review the current projects that are deployed on the web Preview Server. Note: You must enable security for web applications that are deployed on the web Preview Server. You can configure the security in the <code>WebContent/WEB-INF/web.xml</code> file that is contained in the web project that you want to deploy to the server.
Users and Groups	Manage the users and groups that are registered on the web Preview Server. Register new users on the server. Remove users from the server. Register new groups on the server. Add or remove users from registered groups. Remove groups from the server.
Security Roles	Manage the security roles that are registered on the web Preview Server. Register security roles on the server. Remove security roles from the server. Map security roles to users. Remove security roles from users. Map security roles to groups. Remove security roles from groups.

12. Save your changes.

Parent topic: [Testing on a Web Preview Server](#)

Parent topic: [Configuring servers for testing and publishing](#)

Adding data using Java Persistence API (JPA)

Java™ Persistence API (JPA) is a specification for the persistence of Java objects to relational databases. You can use JPA to manage relational data in enterprise applications.

Procedure

1. [Create a web project and activate the Java Persistence facet.](#)
2. [Create JPA entity beans](#) or
3. [Configure JPA entity beans.](#)
 - A. [Configure the primary key](#)
 - B. [Configure named queries](#)
 - C. [Configure relationships](#)
 - D. [Configure concurrency control](#)
 - E. [Configure advanced options](#)

What to do next

For more information about Java Persistence API, see [Developing JPA applications](#).

Parent topic: [Developing web applications](#)

Editing source code

You can use structured text editors to create, modify, and validate your source code.

About this task

A structured text editor is any of several editors that you can use to edit markup languages such as cascading style sheets (CSS), DTD, HTML, JavaScript, JSP, XML, and XSD (schema).

A structured text editor provides capabilities such as find and replace, undo, redo, a spelling checker, and coding assistance. It also highlights syntax in different colors. Structured text editor capabilities include:

- syntax highlighting

- Each keyword type and syntax type is highlighted differently, enabling you to easily find a certain type of keyword for editing. For example, in HTML, element names, attribute names, attribute values, and comments have different colors; in JavaScript, function and variable names, quoted text strings, and comments have different colors.

- unlimited undo and redo

- These options enable you to incrementally undo and redo every change made to a file for the entire editing session. For text, changes are incremented one character or set of selected characters at a time.

- content assist

- Content assist helps you to insert JavaScript functions, HTML tags, or other keywords. Choices available in the content assist list are based on functions that are defined by the syntax of the language in which the file is coded.

- user-defined templates and snippets

- By using the Snippets view, you can access user-defined code snippets and (for all code types except JavaScript) templates to help you quickly add regularly used text strings.

- function selection

- Based on the location of your cursor, the function or tag selection indicator highlights the line numbers that include a function or tag in the vertical ruler on the Source page.

- pop-up menu options

- These are the same editing options available in the workbench **Edit** menu.

Tip: Before you search in a structured text editor, save the file that you are searching. The search function works from the most recently saved version of the file rather than from the contents that you see in the editor area. You do not need to save your file before you find and replace text.

Procedure

1. Determine which editor to use based on the type of file that you want to edit. *Table 1. Editors available for editing source code. Use the editor for the file type.*

File type	Editor	Content assist?
Cascading style sheet	CSS Editor	Yes
Document type definitions	DTD Editor	No
HTML	Rich Page Editor (default)HTML Editor	Yes
JavaScript	Source tab of JavaScript Editor	Yes
JSP	Rich Page Editor (default)JSP Editor	Yes
XML	Source tab of XML Editor	Yes
XSD (schema)	Source tab of XML Schema Editor (default)XML Editor	Yes

2. Open the editor. Right-click a relevant file name in Navigator or Package Explorer view and then click **Open With** and select the editor.

- **File encoding**

You can specify and invoke the character encoding in XML files, XHTML and HTML files, and JSP files in different ways. Because many editors expect to find encoding in source files, specify the encoding in each one of your source files.

- **Content assist**

Content assist helps you insert or finish a tag or function or finish a line of code in a structured text editor. The placement of the cursor in the source file provides the context for the content assist to offer suggestions for completion.

- **Validating source**

You validate source files to determine whether the files contain errors.

- **Editing with snippets**

In the Snippets view, you can catalog and organize reusable programming objects, such as HTML tagging, JavaScript, and JSP code, along with files and custom JSP tags. The view can be extended based on additional objects that you define and include.

- **Configuring structured text editor preferences**

- **Adding and removing markup language templates**

Content assist provides templates, or chunks of predefined code, that you can insert into a file. You can use the default templates as provided, customize the default templates, or create your own templates. Templates are available for the HTML, XML, and JSP markup languages

- **Associating editors with additional files types**

You can associate editors with a file type.

Parent topic: [Developing web applications](#)

File encoding

You can specify and invoke the character encoding in XML files, XHTML and HTML files, and JSP files in different ways. Because many editors expect to find encoding in source files, specify the encoding in each one of your source files.

For JSP files, you might use the `pageEncoding` attribute, the `contentType` attribute, or both attributes in the page directive, as shown in the following example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
```

For XML files, you might use the encoding pseudo-attribute in the `xml` declaration at the start of a document or the text declaration at the start of an entity, as in the following example: `<?xml version="1.0" encoding="iso-8859-1" ?>`

For XHTML and HTML files, you might use the `<meta>` tag inside the `<head>` tags, as shown in the following example:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Parent topic: [Editing source code](#)

Content assist

Content assist helps you insert or finish a tag or function or finish a line of code in a structured text editor. The placement of the cursor in the source file provides the context for the content assist to offer suggestions for completion.

Most of the structured text editors have content assist. For a list of editors that have content assist, refer to [Editing source code](#). For information on configuring content assistance, refer to [Configuring structured text editor preferences](#).

The following sections describe specifics of:

- [HTML content assist](#)
- [JavaScript content assist](#)
- [JSP content assist](#)

HTML content assist

HTML is flexible in that some HTML elements enable end tags to be optionally omitted, such as `P`, `DT`, `DD`, `LI`, `THEAD`, `TR`, `TD`, `TH`, and so on. Other HTML elements that are defined to have no content might require the end tag to always be omitted, such as `BR`, `HR`, `LINK`, `META`, and `IMG`. This flexibility makes the content assist function within the HTML source page editor less precise than it might be with a more rigidly constrained markup language.

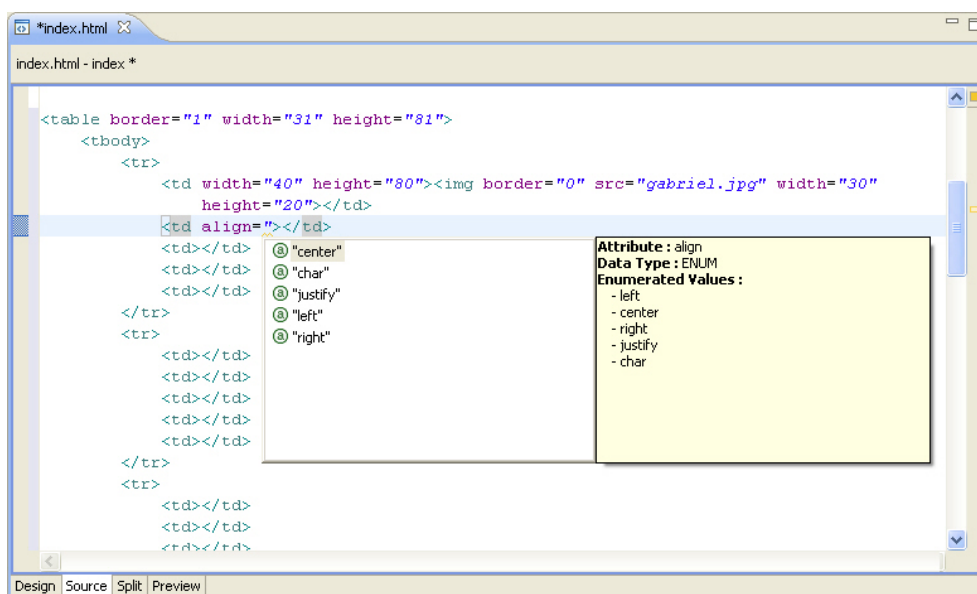
HTML content assist is most beneficial when you need to complete a tag name, add an attribute name-value pair within a start tag, or select from an enumerated list of attribute values.

Although content assist shows only attribute names that are not already specified in a start tag, it does not take into account grammar constraints for tags. For example, the `HEAD` element in HTML permits only zero or one occurrence of a `TITLE` tag in its content. If you prompt for content assist within a `HEAD` content that already contains a `TITLE` tag, content assist still show `TITLE` in its proposal list; however, `TITLE` is de-emphasized in the list.

However, if an attribute is required according to the DTD/Schema, that attribute appears on the list with a yellow circle indicator on its icon.

If your cursor is in a position where content assist is available, a list of available choices is displayed. The list is based on the context. For example, if you use content assist directly after an opening paragraph tag, `<p>`, the first item in the content assist list is the corresponding closing paragraph (`</p>`) tag.


The content assist list displays all available tags for the current cursor position, including templates. The following image shows the default content assist list for a paragraph tag example:



Tag proposals are listed alphabetically. If you type a `<` to begin a new tag before you prompt for content assist, type the first few letters of the tag, the proposal list automatically refreshes to reflect proposals that match the pattern you typed. If

you do not type a < before you prompt for content assist, you can click within the proposal list and then type the first letter in the tag to reduce the amount of scrolling.

As you type the first few letters of the attribute names or enumerated attribute values that you want to add to a tag, the list automatically refreshes to reflect proposals that match the pattern you typed.


Restriction:  When you use Linux (Motif or GTK) and a DBCS locale, double-clicking the content assist list can sometimes cause the Java™ VM to terminate. Instead of double-clicking the list, use the arrows and Enter keys to make the selection from the list.

JavaScript content assist

Code proposals are listed alphabetically. If you type a period followed by a space before you prompt for content assist and begin to type the first few letters of the code, the proposal list automatically refreshes to reflect proposals that match the pattern you typed to reduce the amount of scrolling.

JSP content assist

You have many options for embedding Java and HTML code in your JSP pages by using content assist.

All of the JSP tags are included both in the template list and in XML format (for example, `<jsp:expression>`). To add JSP scriptlet tags, for example, move the cursor to the appropriate position in the file and press Ctrl+Space to use content assist. Select  jsp scriptlet `<%..%>` from the proposal list to insert `<% %>` in the document.

Scriptlets are inserted in a tag `<% %>`. For example: `<% System.currentTimeMillis(); %>`

This example evaluates the Java statement to get the current time in milliseconds.

To insert the result of the statement into the file, put an equals sign (=) in the front of the statement. For example: `This is the time : <%= System.currentTimeMillis(); %>`

When you are within a scriptlet, you are writing pure Java code. Therefore, content assist works the same as it does for the Java editor. For example, if you request content assist after `System`, content assist displays a list of methods. **Note:** Java content assist works only in a web project because it requires a buildpath to find the appropriate Java classes.

There are also special tags such as `useBean`. For example: `<jsp:useBean id="useBean" class="java.lang.String"/>`

Using the `useBean` tag, you can create a `aString` bean of type `String`. When you use content assist, this bean is recognized as a declared variable. For example, if you use content assist after `<% aString. %>`, the content assist list shows available methods because `aString` is declared as a bean of type `String`.

If you use content assist after `<% a %>`, content assist knows that `aString` exists, and it is shown in the content assist list.

Parent topic: [Editing source code](#)

Validating source

You validate source files to determine whether the files contain errors.

About this task

When you work with source files in a structured source editor, your files are validated in two ways:

- [Source validation](#)

- [Batch validation](#)

Procedure

- Validating files when you edit source code. Source validation occurs as you type your code before the file is saved or built.

For example, if you were to type the following code in a JSP editor: `<exa:mp1>`

where `exa:mp1` is a tag that does not exist, the problem is discovered immediately and would appear underlined in the editor. The advantage of this type of validation is that it can alert you to errors instantly.

To turn source validation on or off for all structured source editors:

1. Click **Window > Preferences > General > Editors > Structured Text Editors**.

2. Select (or clear) **Report problems as you type**.

- Validating files during batch processing. Batch validation occurs on saved files. It can identify build process errors and other errors that the source validator cannot identify. For example, in the following JSP code the same prefix is used

twice: `<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>`

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
```

This would prompt the batch validator to trigger markers and to generate build warnings in the Problems view and in the Navigator.

Batch validation can uncover errors in multiple files at once and give you a comprehensive view of where problematic code can be found in your project. Moreover, you do not need to open files in an editor to run batch validation.

- To run batch validation on specific files, select and right-click the files in the Enterprise Explorer and then select **Run Validation** from the menu.

- To set preferences for batch validation, click **Window > Preferences > Validation**.

- [Configuring validation preferences](#)

Parent topic: [Editing source code](#)

Configuring validation preferences

About this task

You can customize your working environment by configuring validation preferences for the structured source editor.

Procedure

1. In the main menu, click **Window > Preferences**.
2. Select one of the choices that are shown in the following table:

Validation preferences	Menu path
HTML	Web > HTML Files > Validation
JavaScript	JavaScript > Validator
JSP	Web > JSP Files > Validation
XML	XML > XML Files > Validation

Parent topic: [Validating source](#)

Editing with snippets

In the Snippets view, you can catalog and organize reusable programming objects, such as HTML tagging, JavaScript, and JSP code, along with files and custom JSP tags. The view can be extended based on additional objects that you define and include.

About this task

The Snippets view has the following features:

- Drag-and-drop to various source editing pages: You can drag items from the view into the active editor and the text is dropped into the document at the cursor location.
- Double-click support: You can double-click an item and have it inserted at the current cursor position in the active editor.
- User-defined drawers and items: You can define, edit, and remove items from view drawers.
- Plug-in-defined drawers and items: Plug-in developers can contribute a default list of items to their own drawers.
- Variables in insertions: By default, items are edited using a dialog and, when inserted, you are prompted for values for each of the variables.
- Customization: You can select which drawers and items are shown in the Snippets view.
- Custom insertion: Plug-in developers can customize the behavior of items so that when they are dropped during a drag-and-drop action, both the text that is inserted and the insertion location are strictly defined.

Procedure

1. Open the Snippets view. Click **Window > Show View > Other > General > Snippets > OK**.
2. Edit your reusable programming objects. In the Snippets view, you can add, edit, or delete snippet items or drawers.

- [Adding snippets drawers](#)
- [Adding items to snippets drawers](#)
- [Editing snippet items](#)
- [Deleting or hiding snippet items or drawers](#)

Parent topic: [Editing source code](#)

Adding snippets drawers

Procedure

1. Right-click anywhere in the Snippets view and select **Customize** from the menu.
2. Click **New > New Category** and type the name of the new drawer in the **Name** field.
3. Optional: Type a description in the **Description** field.
4. Select the drawer's behavior by checking check boxes as appropriate. The check boxes are as follows:

Option	Description
Hide	Do not display the drawer.
Open drawer at start-up	At start, display the drawer's items.
Pin drawer open at start-up	Keep the drawer open.

5. Click **OK**. The new drawer is added to the list of drawers in the Snippets view.

What to do next

After you create the new drawer, [add snippets to the drawer](#).

Parent topic: [Editing with snippets](#)

Adding items to snippets drawers

Procedure

1. Do one of the following choices:

To start with this item:	Do this action:
An empty item that you can edit	Right-click anywhere in an existing drawer, select Customize , and click New > New Item .
Existing text that is pasted into a new item in an existing drawer	Copy or cut text to the clipboard. In the Snippets view, right-click anywhere in an existing drawer and click Paste .
Existing text that is pasted into a new item in a new or existing drawer	Select the text, right-click and then select Add to Snippets . In the New Category dialog box, specify the name of the drawer to which you want to add the item, and click OK .

A Customize Palette window appears.

2. Type a name for the new item, and provide a description. The value that you type in the **Name** field appears next to the item icon in the Snippets view.
3. To include in the **Template Pattern** field, a variable that you already defined, click **Insert Variable Placeholder**. A variable placeholder is a marker that, when tagging is inserted into the active file, is replaced by the value that is entered into the **Insert Template:Item_name** dialog at insertion time. Clicking **Insert Variable Placeholder** or typing Ctrl+Space activates a pop-up in the text area that prompts you to create a marker for the variable. For example, if you create two variables that are named *uri* and *prefix*, and create variable placeholders for both, the template pattern might look like the following example:

```
<%@ taglib uri="${uri}" prefix="${prefix}" %>
```
4. Click **OK**.

Results

The new item is added to the list of items in the appropriate drawer.

Parent topic: [Editing with snippets](#)

Editing snippet items

Procedure

1. In the Snippets view, right-click the name of the item that you want to modify, and select **Customize**.
2. Optional: Type a new name and a new description for the item.
3. To declare a variable, click **New** and type the variable name, description, and default value.
4. To edit an existing variable, type over the existing values.
5. To edit the **Template Pattern** field, type into the field, cut and paste into the field, or if you defined one or more variables click **Insert Variable Placeholder** and double-click the name of the variable that you want to insert. For example, if you have declare variables that are named *uri* and *prefix*, clicking **Insert Variable Placeholder** brings up a menu that contains those names. Double-clicking *uri* inserts `${uri}`, as in the following example:

```
<%@ taglib uri="${uri}"
```

```
prefix="${prefix}" %>
```

Later, when you insert the snippet into a file, `${uri}` and `${prefix}` are each converted to the default value declared in the Variables table. Users can replace the default values in the **Insert Template**:*Item_name* dialog at insertion time.

6. Click **OK**.

Results

The modified item is added to the list of items in the appropriate drawer.

Parent topic: [Editing with snippets](#)

Deleting or hiding snippet items or drawers

About this task

You can hide any drawer or item that shows up in the Snippets view. If the drawer or item is user-defined, you can delete it permanently. To unhide the drawer, right-click anywhere in the Snippets view and click **Customize**. In the Customize Palette dialog, select your drawer and clear **Hide**. To delete or hide a drawer or item, complete the following steps:

Procedure

1. Select the item or drawer that you want to remove.
2. Right-click and select **Customize** from the menu.
3. Select the item or drawer and click **Delete** or **Hide**.
4. Click **OK**. The item or drawer is deleted or hidden from the Snippets view.

Parent topic: [Editing with snippets](#)

Configuring structured text editor preferences

About this task

You can customize your working environment by configuring preferences for the structured text editor.

Procedure

1. In the main menu, click **Window > Preferences**.
2. Select one of the choices that are shown in the following table:

Item	Menu path
Annotation settings	General > Editors > Text Editors > Annotations
Character encoding	Web , then one of the following choices: CSS FilesHTML FilesJSP FilesXML > XML Files
Content assist: HTML	Web > HTML Files > Editor
Content assist: JavaScript	JavaScript > Editor > Content Assist
Content assist: XML	XML > XML Files > Editor
Editor appearance	General > Editors > Structured Text Editors > Appearance
Editor font	General > Appearance > Colors and Fonts
Editor navigation	General > Editors > Text Editors > Hyperlinking
File-type-specific settings	Web , then navigate to the file type and particular setting
Hover help	General > Editors > Structured Text Editors > Hovers
Keyboard shortcuts	General > Keys

Parent topic:[Editing source code](#)

Adding and removing markup language templates

Content assist provides templates, or chunks of predefined code, that you can insert into a file. You can use the default templates as provided, customize the default templates, or create your own templates. Templates are available for the HTML, XML, and JSP markup languages

- **Adding and removing HTML templates**

HTML content assist provides several templates, or chunks of predefined code, that you can insert into a file. You can use the default templates as provided, customize the default templates, or create your own templates.

- **Adding and removing JSP templates**

JSP content assist provides several templates, or chunks of predefined code, that you can insert into a file. You can use the default templates as provided, customize the default templates, or create your own templates.

- **Adding and removing XML templates**

XML content assist provides a comment template, a chunk of predefined code that you can insert into a file. You can use the default template as provided, customize that template, or create your own templates.

Parent topic:[Editing source code](#)

Adding and removing HTML templates

HTML content assist provides several templates, or chunks of predefined code, that you can insert into a file. You can use the default templates as provided, customize the default templates, or create your own templates.

About this task

For example, you can work on a group of HTML pages that all contain a table with a specific appearance. Create a template that contains the tags for that table, including the appropriate attributes and attribute values for each tag. You can copy and paste the tags from a structured text editor into the **Pattern** field of the template. Then, select the name of the template from a content assist proposal list whenever you want to insert your custom table into an HTML or XHTML file.

To add an HTML template, complete the following steps:

Procedure

1. From the **Window** menu, select **Preferences**.
2. In the Preferences page, select **Web > HTML Files > Editor > Templates**.
3. Click **New**.
4. Enter the new template name (a text string) and a brief description of the template.
5. Using the **Context** list, specify the context in which the template is available in the proposal list when content assist is requested.
6. In the **Pattern** field, enter the appropriate tags, attributes, or attribute values (the content of the template) to be inserted by content assist.
7. If you want to insert a variable, click **Insert Variable** and select the variable to be inserted. For example, the *word_selection* variable indicates the word that is selected at the beginning of template insertion, and the *cursor* variable determines where the cursor will be after the template is inserted in the HTML document.
8. Click **OK** to save the new template.

What to do next

You can edit, remove, import, or export a template by using the same Preferences page. If you modified a default template, you can restore it to its default value. If you have not exited from the workbench since it was removed, you can also restore a removed template.

If you have a template that you do not want to remove but you no longer want the template to appear in the content assist list, go to the Templates preferences page and clear its check box.

Parent topic: [Adding and removing markup language templates](#)

Adding and removing JSP templates

JSP content assist provides several templates, or chunks of predefined code, that you can insert into a file. You can use the default templates as provided, customize the default templates, or create your own templates.

About this task

For example, you can work on a group of JSP pages that all contain a table with a specific appearance. Create a template that contains the tags for that table, including the appropriate attributes and attribute values for each tag. You can copy and paste the tags from a structured text editor into the **Pattern** field of the template. Then, select the name of the template from a content assist proposal list whenever you want to insert your custom table into a JSP file.

To add a JSP template, complete the following steps:

Procedure

1. From the **Window** menu, select **Preferences**.
2. In the Preferences page, select **Web > JSP Files > Editor > Templates**.
3. Click **New**.
4. Enter the new template name (a text string) and a brief description of the template.
5. Using the **Context** list, specify the context in which the template is available in the proposal list when content assist is requested.
6. In the **Pattern** field, enter the appropriate tags, attributes, or attribute values (the content of the template) to be inserted by content assist.
7. If you want to insert a variable, click **Variable** and select the variable to be inserted. For example, the *word_selection* variable indicates the word that is selected at the beginning of template insertion. As another example, the *cursor* variable determines where the cursor will be after the template is inserted in the HTML document.
8. Click **OK** to save the new template.

What to do next

You can edit, remove, import, or export a template by using the same Preferences page. If you modified a default template, you can restore it to its default value. If you have not exited from the workbench since it was removed, you can also restore a removed template.

If you have a template that you do not want to remove but you no longer want the template to appear in the content assist list, go to the Templates preferences page and clear its check box.

Parent topic: [Adding and removing markup language templates](#)

Adding and removing XML templates

XML content assist provides a comment template, a chunk of predefined code that you can insert into a file. You can use the default template as provided, customize that template, or create your own templates.

About this task

For example, you can work on a group of XML pages that all contain a table with a specific appearance. Create a template that contains the tags for that table, including the appropriate attributes and attribute values for each tag. You can copy and paste the tags from a structured text editor into the **Pattern** field of the template. Then, select the name of the template from a content assist proposal list whenever you want to insert your custom table into an XML file.

To add an XML template, complete the following steps:

Procedure

1. From the **Window** menu, select **Preferences**.
2. In the Preferences page, select **XML > XML Files > Editor > Templates**.
3. Click **New**.
4. Enter the new template name (a text string) and a brief description of the template.
5. Using the **Context** drop-down list, specify the context in which the template is available in the proposal list when content assist is requested.
6. In the **Pattern** field, enter the appropriate tags, attributes, or attribute values (the content of the template) to be inserted by content assist.
7. If you want to insert a variable, click the **Variable** button and select the variable to be inserted. For example, the `word_selection` variable indicates the word that is selected at the beginning of template insertion. For another example, the `cursor` variable determines where the cursor will be after the template is inserted in the XML document.
8. Click **OK** to save the new template.

What to do next

You can edit, remove, import, or export a template by using the same Preferences page. If you modified a default template, you can restore it to its default value. If you have not exited from the workbench since it was removed, you can also restore a removed template.

If you have a template that you do not want to remove but you no longer want the template to appear in the content assist list, go to the Templates preferences page and clear its check box.

Parent topic: [Adding and removing markup language templates](#)

Associating editors with additional files types

You can associate editors with a file type.

Procedure

1. Associate an editor with a file type.
 - A. Click **Window > Preferences > General > Editors > File Associations**.
 - B. Configure the associations in the File Associations window.
2. If you cannot complete step 1 because your Eclipse-based editor recognizes only a file that is based on the contents of the file, you might need to first map the content type with its file extension before you can associate an additional file type with the editor.
 - A. Map a content type with a file extension.
 1. Click **Window > Preferences > General > Content Types**. The Content Type window displays.
 2. Select a content type in the Content Type window, and then click **Add**. The Define a New File Type window displays.
 3. Type the name of the file type, and then click **OK**.
 - B. Complete step 1 to associate an editor with a file type.

Parent topic: [Editing source code](#)

Web application tutorials, samples, and videos


To view the samples and tutorials in this product, click **Help > Help Contents** and expand the Samples and Tutorials sections. To view the videos, click the links. Learn about different aspects of web application development from the following videos, samples, and tutorials:

Note: These video tutorials demonstrate the use of Rational® Application Developer, however, they are also useful for users of WebSphere® Developer Tools.

-  [Tutorials](#)

Parent topic: [Developing web applications](#)

Tutorials

-  **Tutorial: Create a loan payment calculator with Dojo**
 - This tutorial demonstrates how to create a Dojo-enabled loan calculator application. You learn how to create a user interface with Dojo widgets; use content assist, templates, and wizards to write Dojo code; and debug your application by using Firebug.

Developing web service applications

You can develop and publish web service applications, which are modular applications that implement a services oriented architecture (SOA). These topics explain how to create and deploy web services, how to implement web service security, and how to test and validate web services.

- [Learn about web service applications](#)

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a services oriented architecture (SOA), which supports the connecting or sharing of resources and data in a very flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

- [SOAP](#)

SOAP (formerly known as Simple Object Access Protocol) is a lightweight protocol for the exchange of information in a decentralized, distributed environment. A SOAP message is a transmission of information from a sender to a receiver. SOAP messages can be combined to perform request/response patterns.

- [Java API for XML based web services](#)

Java™ API for XML-based web services (JAX-WS), which is also known as JSR-224, is the next generation web services programming model that extends the foundation provided by the Java API for XML-based RPC (JAX-RPC) programming model. Using JAX-WS, developing web services and clients is simplified with greater platform independence for Java applications by the use of dynamic proxies and Java annotations. The web services tools included in this product support JAX-WS 2.0, 2.1, and 2.2.

- [JAXB](#)

Java Architecture for XML Binding (JAXB), which is also known as JSR-222, is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of web services. JAXB leverages the flexibility of platform-neutral XML data in Java applications to bind XML schema to Java applications without requiring extensive knowledge of XML programming. The tools included in this workbench implement JAXB 2.0, 2.1, and 2.2 standards.

- [Developing JAX-RS applications](#)

You can develop Java API for RESTful web services (JAX-RS) applications so that you can create Representational State Transfer (REST) services quickly.

- [Tools for web services development](#)

- [Configuring a workspace for web services development](#)

Although you can begin web services development immediately upon creating a workspace, you might find it convenient to configure your workspace to optimize your development experience.

- [Developing Web services and clients](#)

You can create web services and clients by using the web services wizards, annotations, Ant tasks, or command-line tools.

- [Web services: Editing, assembling, and securing tasks](#)

After you create a web service or client, you can do various assembly tasks, such as editing the web service deployment descriptors, adding handlers, and enabling security.

- [Creating and editing JAX-WS web service handlers](#)

You can add JAX-WS logical or protocol handlers to intercept inbound and outbound messages to or from web services and their clients. You can select from any currently available JAX-WS web services and clients and start the Handler Creation Wizard. In the wizard, you provide the class name of the handler, the handler name, and an optional display name, and specify the type of handler. When finished, the wizard generates the skeleton handler code and updates the applicable deployment descriptor.

- [Merging web services updates](#)

After you create a web service, you might want to change it. Although you cannot automatically propagate all your

changes to all the required files, to retain your changes while you update the web service, you can merge a generated skeleton file. You can then regenerate your web service, and your changes remain intact.

- **Securing web services**

Web services security for WebSphere® Application Server is based on the OASIS web services security (WSS) Version 1.0 specification, the Username token Version 1.0 profile, and the X.509 token Version 1.0 profile. These standards and profiles address how to provide protection for messages that are exchanged in a web service environment.

- **Deploying web services**

Deploying a web service involves creating the code that makes your web service available to others. You can deploy a project, an EAR file, or an application client. If you created a web service by using the web services wizards, the deployment code is generated automatically.

- **Testing and validating web services**

After you create a web service or client, you can test it using sample JSPs, the web services Explorer, or the Generic Service Client. You can also test the SOAP traffic that is passed by the service.

- **Limitations of web services**

This file contains a comprehensive list of limitations, both permanent and temporary, that affect web services.

Related concepts:




[↗ JSR 109 - Implementing enterprise web services](#)

Learn about web service applications

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a services oriented architecture (SOA), which supports the connecting or sharing of resources and data in a very flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

Overview

You can read the following topics before creating a web service. They provide planning and technology overview information that might be useful if you are new to web services or developing web services in this development environment.

-  [Overview of web services](#)
-  [JAX-WS \(JSR-224\)](#)
-  [SOAP](#)




Getting Started

If you are already familiar with web services technology the following topics will help you set up your workspace for web services development, and guide you through the development process.

-  [Optimizing the workspace for web services development](#)
-  [Creating a JAX-WS top-down web service](#)
-  [Creating a JAX-WS bottom-up web service](#)
-  [Creating a JAX-WS web service using annotations](#)

Samples and Tutorials

These web services samples and tutorials are included with this product:

-  [Sample: WebSphere® JAX-WS web service temperature conversion](#)
 - This sample creates a Java EE 5 web service and web service client created from an EJB 3.0 Enterprise bean that provides methods to convert Celsius to Fahrenheit and Fahrenheit to Celsius. It uses the WebSphere JAX-WS runtime environment and runs on WebSphere Application Server v7.0.
-  [Tutorial: Creating a secured JAX-WS web service from an WSDL file](#)
 - This tutorial walks you through the steps to create a JAX-WS web service and client, and to secure it using a policy set. The tutorial ends with the generation of a code similar to that in the JAX-WS RSP address book sample.
-  [Tutorial: Creating a JAX-RS web service](#)
 - This tutorial walks you through creating a JAX-RS application.

Web resources for learning

In addition to the information found in this information center, the following links provide learning material.

[IBM Redbooks®: Rational® Application Developer for WebSphere Software V8 Programming Guide - Draft](#)

[IBM Redbooks: Rational Application Developer V7.5 Programming Guide](#). Chapter 18 focuses on web services.

[IBM Redbooks: Rational Application Developer V7 Programming Guide](#). Chapter 18 focuses on web services.

[IBM Redbooks: web services Feature Pack for WebSphere Application Server V6.1](#)

[Overview of web services](#)

[Web Services Description Language \(WSDL\)](#)

[Universal Description, Discovery, and Integration \(UDDI\)](#)

[Web Services Inspection Language \(WSIL\)](#)

Note: [Latest developerWorks® articles and tutorials about web services](#)

Parent topic: [Developing web service applications](#)

SOAP

SOAP (formerly known as Simple Object Access Protocol) is a lightweight protocol for the exchange of information in a decentralized, distributed environment. A SOAP message is a transmission of information from a sender to a receiver.

SOAP messages can be combined to perform request/response patterns.

SOAP is transport independent but is most commonly carried over HTTP in order to run with the existing Internet infrastructure. SOAP enables the binding and use of discovered web services by defining a message path for routing messages. SOAP is used to query UDDI for web services.

The Java™ API for XML web services (JAX-WS) standard introduces the ability to support both SOAP 1.1 as well as SOAP 1.2. With WebSphere® Application Server V7.0 and later or WebSphere Application Server Liberty profile installed, the workbench supports SOAP 1.1 and SOAP 1.2.

SOAP 1.1

SOAP 1.1 is a protocol-independent transport and can be used in combination with a variety of protocols. In web services that are developed and implemented with WebSphere Application Server, SOAP is used in combination with HTTP, HTTP extension framework, and Java Message Service (JMS). SOAP is also operating-system independent and not tied to any programming language or component technology. Provided that the client can issue XML messages, it does not matter what technology is used to implement the client. Similarly, the service can be implemented in any language, provided that the service can process SOAP messages. Also, both server and client sides can reside on any suitable platform.

SOAP is an XML-based protocol that defines three parts of every message:

- **Envelope.** The envelope defines a framework for describing what is in a message and how to process it. A SOAP message is an envelope that contains zero or more headers and exactly one body. The envelope is the top-level element of the XML document, providing a container for control information, the address of a message, and the message itself. Headers transport any control information such as quality-of-service attributes. The body contains the message identification and its parameters. Both the headers and the body are child elements of the envelope.
- **Encoding rules.** The set of encoding rules expresses instances of application-defined data types. Encoding rules define a serialization mechanism that can be used to exchange instances of application-defined data types. SOAP defines a programming language-independent data type scheme based on XSD plus encoding rules for all data types defined according to this model. SOAP encoding is not WS-I compliant and thus the Literal use (which is no encoding) is suggested for interoperable Web services and required for WS-I compliance.
- **Communication styles.** Communications can follow a remote procedure call (RPC) or message-oriented (Document) format.

SOAP supports two different communication styles:

- **Remote procedure call (RPC):** RPC is the invocation of an operation returning a result. RPC is typically used with SOAP encoding, which is not WS-I compliant.
- **Document Style:** Document style is also known as document-oriented or message-oriented style. This style provides a deeper layer of abstraction, and requires more programming work.

In distributed computing environments, encoding styles define how data values that are defined in the application can be translated to and from a particular protocol format. The translation process is known as serialization and deserialization.

The SOAP 1.1 specification defines the SOAP encoding style:

- **SOAP encoding:** The SOAP encoding style enables you to serialize/deserialize values of data types from the SOAP data model. This encoding style is defined in the SOAP 1.1 standard, and is not WS-I compliant.

WSDL defines the Literal XML encoding style:

- **Literal XML:** Literal refers to the fact that the document should be read as-is, or unencoded. The document is serialized as XML, meaning that the message XML complies with the Schema in the WSDL. When using Literal encoding, each message part references a concrete schema definition. Literal encoding is WS-I compliant.

SOAP 1.2

The SOAP 1.2 specification is also a World Wide Web Consortium (W3C) recommendation, and the tools included in this workbench follow the standards that are outlined in SOAP 1.2. The SOAP 1.2 specification comes in three parts plus some assertions and a test collection:

- [Part 0: Primer](#)
- [Part 1: Messaging Framework](#)
- [Part 2: Adjuncts](#)
- [Specification Assertions and Test Collection](#)

SOAP 1.2 provides a more specific definition of the SOAP processing model, which removes many of the ambiguities that sometimes led to interoperability problems in the absence of the Web Services-Interoperability (WS-I) profiles. SOAP 1.2 should reduce the chances of interoperability issues with SOAP 1.2 implementations between different vendors.

Some of the more significant changes in the SOAP 1.2 specification include:

- The ability to now officially define other transport protocols other than the HTTP protocol as long as vendors conform to the binding framework that is defined in SOAP 1.2. While HTTP is ubiquitous, it is not as reliable of a transport as other things such as TCP/IP, MQ, and so forth.
- The fact that SOAP 1.2 is based on the XML Information Set (XML Infoset). The information set provides a way to describe the XML document using the XSD schema but does not necessarily serialize the document by using XML 1.0 serialization. SOAP 1.1 is based upon XML 1.0 serialization. The information set will make it easier to use other serialization formats such as a binary protocol format. You can use a binary protocol format shrink the message into a much more compact format where some of the verbose tagging information might not be required.

Additional information about SOAP

For a more detailed examination of the differences between SOAP 1.1 and 1.2 refer to: [Differences in SOAP versions](#)

Note: [Latest developerWorks® articles and tutorials about SOAP and Web services](#)

- **SOAP MTOM**

SOAP Message Transmission Optimization Mechanism (MTOM) is a standard that is developed by the World Wide Web Consortium (W3C). MTOM describes a mechanism for optimizing the transmission or wire format of a SOAP message by selectively re-encoding portions of the message while still presenting an XML Information Set (Infoset) to the SOAP application.

Parent topic: [Developing web service applications](#)

Related concepts:

[SOAP MTOM](#)

[Java API for XML based web services](#)

[JAXB](#)

[JSR 109 - Implementing enterprise web services](#)

SOAP MTOM

SOAP Message Transmission Optimization Mechanism (MTOM) is a standard that is developed by the World Wide Web Consortium (W3C). MTOM describes a mechanism for optimizing the transmission or wire format of a SOAP message by selectively re-encoding portions of the message while still presenting an XML Information Set (Infoset) to the SOAP application.

MTOM uses XML-binary Optimized Packaging (XOP) in the context of SOAP and MIME over HTTP. XOP defines a serialization mechanism for the XML Infoset with binary content that is not only applicable to SOAP and MIME packaging, but to any XML Infoset and any packaging mechanism. It is an alternate serialization of XML that just happens to look like a MIME multipart or related package, with XML documents as the root part. That root part is very similar to the XML serialization of the document, except that base64-encoded data is replaced by a reference to one of the MIME parts, which is not base64 encoded. This reference enables you to avoid the bulk and overhead in processing that is associated with encoding. Encoding is the only way binary data can work directly with XML.

If MTOM mapping generation is disabled, then XOP is disabled. If XOP is disabled, the binary data are not sent by using MIME attachments. Instead, the binary data is base64 encoded as usual.

Without MTOM, the data is encoded in whatever format is described in the schema (base64 or hex) and then is displayed in the XML document. The following example shows a SOAP message with an `<xsd:base64Binary>` element:

```
... other transport headers ...
Content-Type: text/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header/>
    <soapenv:Body>
      <sendImage xmlns="http://org.apache.axis2/jaxws/sample/mtom">
        <input>
          <imageData>R0lGODl ... more base64 encoded data ... KTJk8giAAA7</imageData>
        </input>
      </sendImage>
    </soapenv:Body>
  </soapenv:Envelope>
```

When MTOM is enabled, the binary data that represents the attachment is included as a MIME attachment to the SOAP message. The following example shows an MTOM-enabled SOAP message with attachment data:

```
... other transport headers ...
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812;
type="application/xop+xml"; start="<0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>";
start-info="text/xml"; charset=UTF-8

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: application/xop+xml; charset=UTF-8; type="text/xml";
content-transfer-encoding: binary
content-id:
  <0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>

  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header/>
```

```
<soapenv:Body>
  <ati:sendImage xmlns="http://org.apache.axis2/jaxws/sample/mtom"
xmlns:ati="http://org.apache.axis2/jaxws/sample/mtom">
  <ati:input>
    <ati:imageData>
      <xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/>
    </ati:imageData>
  </ati:input>
</ati:sendImage>
</soapenv:Body>
</soapenv:Envelope>
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: text/plain
content-transfer-encoding: binary
content-id:
  <1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org>
```

... binary data goes here ...

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812--

For more information, refer to the official MTOM specification: [SOAP Message Transmission Optimization Mechanism](#)

Parent topic: [SOAP](#)

Related concepts:

[SOAP](#)

[Java API for XML based web services](#)

[JAXB](#)

[JSR 109 - Implementing enterprise web services](#)

Java API for XML based web services

Java™ API for XML-based web services (JAX-WS), which is also known as JSR-224, is the next generation web services programming model that extends the foundation provided by the Java API for XML-based RPC (JAX-RPC) programming model. Using JAX-WS, developing web services and clients is simplified with greater platform independence for Java applications by the use of dynamic proxies and Java annotations. The web services tools included in this product support JAX-WS 2.0, 2.1, and 2.2.

JAX-WS is a new programming model that simplifies application development through support of a standard, annotation-based model to develop web service applications and clients. The JAX-WS programming standard strategically aligns itself with the current industry trend toward a more document-centric messaging model and replaces the remote procedure call programming model as defined by JAX-RPC. Although this product still supports the JAX-RPC programming model and applications, JAX-RPC has limitations and does not support many current document-centric services. JAX-WS is the strategic programming model for developing web services and is a required part of the Java EE 5 platform.

Implementing the JAX-WS programming standard provides the following enhancements for developing web services and clients:

- Better platform independence for Java applications

- Using JAX-WS APIs, developing web services and clients is simplified with better platform independence for Java applications. JAX-WS takes advantage of dynamic proxies whereas JAX-RPC uses generated stubs. The dynamic proxy client invokes a web service based on a Service Endpoint Interface (SEI) which is generated or provided. The dynamic proxy client is similar to the stub client in the JAX-RPC programming model. Although the JAX-WS dynamic proxy client and the JAX-RPC stub client are both based on the Service Endpoint Interface (SEI) that is generated from a WSDL file, there is a major difference. The dynamic proxy client is dynamically generated at run time using the Java 5 dynamic proxy functionality, while the JAX-RPC-based stub client is a non-portable Java file that is generated by tooling. Unlike the JAX-RPC stub clients, the dynamic proxy client does not require you to regenerate a stub prior to running the client on an application server for a different vendor because the generated interface does not require the specific vendor information. Refer to Chapter 4 of the JAX-WS 2.0 specification for more information on using dynamic proxy clients.

- Annotations

- JAX-WS introduces support for annotating Java classes with metadata to indicate that the Java class is a web service. JAX-WS supports the use of annotations based on the Metadata Facility for the Java Programming Language (JSR 175) specification, the web services Metadata for the Java Platform (JSR 181) specification and annotations that are defined by the JAX-WS 2.0 specification. Using annotations in the Java source and in the Java class simplifies development of web services by defining some of the additional information that is typically obtained from deployment descriptor files, WSDL files, or mapping metadata from XML and WSDL files into the source artifacts.

For example, you can embed a simple `@WebService` tag in the Java source to expose the bean as a web service.

```
@WebService

public class QuoteBean implements StockQuote {

    public float getQuote(String sym) { ... }

}
```

The annotation `@WebService` tells the server runtime environment to expose all public methods on that bean as a web service. Additional levels of granularity can be controlled by adding additional annotations on individual methods or parameters. Using annotations makes it much easier to expose Java artifacts as web services. In addition, as artifacts are created from using some of the top-down mapping tools starting from a WSDL file, annotations are

included within the source and Java classes as a way of capturing the metadata along with the source files.

- Invoking web services asynchronously

- With JAX-WS, web services can be called both synchronously and asynchronously. JAX-WS adds support for both a polling mechanism and callback mechanism when calling web services asynchronously. Using a polling model, a client can issue a request, get a response object back, which is polled to determine whether the server has responded. When the server responds, the actual response is retrieved. Using the polling model, the client can continue to process other work without waiting for a response to return. Using the callback model, the client provides a callback handler to accept and process the inbound response object. Both the polling and callback models enable the client to focus on continuing to process work while providing for a more dynamic and efficient model to invoke web services.

For example, a web service interface has methods for both synchronous and asynchronous requests.

```
@WebService
public interface CreditRatingService {
    // sync operation
    Score    getCreditScore(Customer customer);
    // async operation with polling
    Response<Score> getCreditScoreAsync(Customer customer);
    // async operation with callback
    Future<?> getCreditScoreAsync(Customer customer,
        AsyncHandler<Score> handler);
}
```

The asynchronous invocation that uses the callback mechanism requires an additional input by the client programmer. The callback handler is an object that contains the application code that will be executed when an asynchronous response is received. The following is a code example for an asynchronous callback handler:

```
CreditRatingService svc = ...;

Future<?> invocation = svc.getCreditScoreAsync(customerFred,
    new AsyncHandler<Score>() {
        public void handleResponse (
            Response<Score> response)
        {
            Score score = response.get();
            // do work here...
        }
    });
```

The following is a code example for an asynchronous polling client:

```
CreditRatingService svc = ...;
Response<Score> response = svc.getCreditScoreAsync(customerFred);

while (!response.isDone()) {
    // do something while we wait
}

// no cast needed, thanks to generics
Score score = response.get();
</Score>
```

- Using resource injection

- JAX-WS supports resource injection to further simplify development of web services. JAX-WS uses this key feature of Java EE 5 to shift the burden of creating and initializing common resources in a Java runtime environment from your web service application to the application container environment, itself. JAX-WS provides support for a subset of annotations that are defined in JSR-250 for resource injection and application life cycle in its runtime environment. The application server also supports the usage of the `@Resource` or `@WebServiceRef` annotation to declare JAX-WS managed clients and to request injection of JAX-WS services and ports. When either of these annotations are used on a field or method, they result in injection of a JAX-WS service or port instance. The usage of these annotations also results in the type specified by the annotation being bound into the JNDI namespace.

The `@Resource` annotation is defined by the JSR-250, Common Annotations specification that is included in Java Platform, Enterprise Edition 5 (Java EE 5). By placing the `@Resource` annotation on a variable of type `javax.xml.ws.WebServiceContext` within a service endpoint implementation class, you can request a resource injection and collect the `javax.xml.ws.WebServiceContext` interface related to that particular endpoint invocation. From the `WebServiceContext` interface, you can collect the `MessageContext` for the request associated with the particular method call using the `getMessageContext()` method.

The following example illustrates using the `@Resource` and `@WebServiceRef` annotations for resource injection:

```
@WebService
public class MyService {

    @Resource
    private WebServiceContext ctx;

    @Resource
    private SampleService svc;

    @WebServiceRef
    private SamplePort port;

    public String echo (String input) {
        ...
    }
}
```

Refer to sections 5.2.1 and 5.3 of the JAX-WS specification for more information on resource injection.

- Data binding with JAXB 2.2

- JAX-WS leverages the JAXB API and tools as the binding technology for mappings between Java objects and XML documents. JAX-WS tooling relies on JAXB tooling for default data binding for two-way mappings between Java objects and XML documents. JAXB data binding replaces the data binding described by the JAX-RPC specification. WebSphere® Application Server Version 7.0 supports the JAXB 2.1 specification. JAX-WS 2.1 requires JAXB 2.1 for data binding. JAXB 2.1 provides enhancements such as improved compilation support and support for the `@XMLSeeAlso` annotation, and full schema 1.0 support.

WebSphere Application Server Version 8.0 or later, and WebSphere Application Server Liberty Profile V8.5.5 supports the JAXB 2.2 specification. JAX-WS 2.2 requires JAXB 2.2 for data binding. JAXB 2.2 provides minor enhancements to its annotations for improved schema generation and better integration with JAX-WS.

- Dynamic and static clients

- The dynamic client programming API for JAX-WS is called the dispatch client (`javax.xml.ws.Dispatch`). The dispatch client is an XML messaging oriented client. The data is sent in either `PAYLOAD` or `MESSAGE` mode. When using

the PAYLOAD mode, the dispatch client is only responsible for providing the contents of the <soap:Body> element and JAX-WS adds the <soap:Envelope> and <soap:Header> elements. When using the MESSAGE mode, the dispatch client is responsible for providing the entire SOAP envelope including the <soap:Envelope>, <soap:Header>, and <soap:Body> elements and JAX-WS does not add anything additional to the message. The dispatch client supports asynchronous invocations using a callback or polling mechanism.

</soap:Body></soap:Header></soap:Envelope></soap:Header></soap:Envelope></soap:Body>

The static client programming model for JAX-WS is the called the proxy client. The proxy client invokes a web service based on a Service Endpoint interface (SEI) which is generated or provided.

- MTOM support

- Using JAX-WS, you can send binary attachments such as images or files along with web services requests. JAX-WS adds support for optimized transmission of binary data as specified by Message Transmission Optimization Mechanism (MTOM).

- Multiple payload structures

- JAX-WS exposes the following binding technologies to the user: XML Source, SOAP Attachments API for Java (SAAJ) 1.3, and Java Architecture for XML Binding (JAXB) 2.0. XML Source enables a user to pass a javax.xml.transform.Source into the runtime which represents the data in a Source object to be passed to the runtime. SAAJ 1.3 now has the ability to pass an entire SOAP document across the interface rather than just the payload itself. This is done by the client passing the SAAJ SOAPMessage object across the interface. JAX-WS leverages the JAXB 2.0 support as the data binding technology of choice between Java and XML.

- SOAP 1.2 support

- Support for SOAP 1.2 was added to JAX-WS 2.0. JAX-WS supports both SOAP 1.1 and SOAP 1.2. SOAP 1.2 provides a more specific definition of the SOAP processing model, which removes many of the ambiguities that sometimes led to interoperability problems in the absence of the web services-Interoperability (WS-I) profiles. SOAP 1.2 should reduce the chances of interoperability issues with SOAP 1.2 implementations between different vendors. It is not interoperable with earlier versions.

- Support for method parameters and return types

- JAX-WS 2.2 supports method parameters and return types. In a JAX-WS web services operation, you can define a web services operation with an operation parameter and an optional return type. If the operation parameter and return type define an empty targetNamespace property by specifying a "" value for the targetNamespace property with either the @WebParam or @WebResult annotation, the JAX-WS runtime environment behaves in the following way:
 - If the operation is document style, the parameter style is WRAPPED and the parameter does not map to a header. An empty namespace is mapped with the operation parameters and return types.
 - If the parameter style is not WRAPPED, the value of the targetNamespace parameter specified using the @WebParam or @WebResult annotation is used.

JAX-WS 2.1.6

When a JAX-WS web service is created from a Java class, the class's public methods are exposed as operations and become part of the web service's WSDL contract. The mapping between these methods and operations is governed mainly by JSR-181 and JSR-250. With these rules, a public method in a Java class and its hierarchy up to but excluding java.lang.Object are exposed when the following conditions are true:

- The method is annotated with @WebMethod or @WebMethod(exclude=false) and the containing class has an @WebService annotation
- The method has no @WebMethod annotation but the containing class has an @WebService annotation and no other methods are annotated with @WebMethod or @WebMethod(exclude=false)

JAX-WS 2.1.6 alters the rules for how a Java class's methods are exposed on the web service interface. Now, a public method in a Java class and its hierarchy up to but excluding java.lang.Object are exposed when the following conditions

are true:

- The method has an `@WebMethod` or `@WebMethod(exclude=false)` annotation.
- The method has no `@WebMethod` annotation but the containing class has a `@WebService` annotation.

Example:

```
public class Base
{
    @WebMethod(exclude=false)
    public void superExposed(String s) {}
    public String supernoanno(String s) {}
}
```

`@WebService`

```
public class BeanImpl extends Base
```

```
{
    @WebMethod(exclude=false)
    public void exposed(String s) {}
    public String nonpublic(String s) {}
}
```

Before JAX-WS 2.1.6, the only exposed method would be `public void exposed(String s)`. In JAX-WS 2.1.6 and later the following methods are exposed:

```
public void exposed(String s)
public String nonpublic(String s)
public void superExposed(String s)
```

WebSphere Application Server Version 7.0.0.7 and later include these changes through IBM® JDK 6 SR6. To enable the workbench to provide guidance on using JAX-WS 2.1.6, go to **Window > Preferences > General > Service policies > WebSphere Programming Models > JAX-WS** and set the JAX-WS 2.1.6 method exposure guidance setting to **true**.

JAX-WS 2.2

The server runtime environments that support JAX-WS Version 2.2 and web services for Java EE (JSR 109) Version 1.3 specifications are:

- WebSphere Application Server Liberty Profile V8.5.5
- WebSphere Application Server V8.5
- WebSphere Application Server V8.0

The JAX-WS 2.2 specification supercedes and includes functions within the JAX-WS 2.1 specification. JAX-WS 2.2 adds client-side support for using `WebServiceFeature`-related annotations such as `@MTOM`, `@Addressing`, and the `@RespectBinding` annotations. JAX-WS 2.1 had previously added support for these annotations on the server. In addition, the web services for Java EE 1.3 specification introduces support for these `WebServiceFeature`-related annotations, as well as support for using deployment descriptor elements to configure these features on both the client and server. JAX-WS 2.2 requires Java Architecture for XML Binding (JAXB) Version 2.2 for data binding.

For more information on JAX-WS, refer to the official JSR-224 specification: [JSR 224: Java API for XML-Based web services \(JAX-WS\) 2.0](#)

Note: [Latest articles and tutorials from developerWorks about JAX-WS](#)

Parent topic: [Developing web service applications](#)

Related concepts:

[JAXB](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Web services runtime environments](#)

[SOAP](#)

[SOAP MTOM](#)

[🔗 JSR 109 - Implementing enterprise web services](#)

[Annotations for creating web services overview](#)

Related tasks:

[Creating a JAX-WS-enabled WebSphere server](#)

[Creating a JAX-WS enabled web project](#)

[Creating a Java bean skeleton from a WSDL document using the WebSphere JAX-WS runtime environment](#)

[Creating a web service from a Java bean using the IBM WebSphere JAX-WS runtime environment](#)

[Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment](#)

[Setting web services preferences](#)

JAXB

Java™ Architecture for XML Binding (JAXB), which is also known as JSR-222, is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of web services. JAXB leverages the flexibility of platform-neutral XML data in Java applications to bind XML schema to Java applications without requiring extensive knowledge of XML programming. The tools included in this workbench implement JAXB 2.0, 2.1, and 2.2 standards.

JAXB is an XML to Java binding technology that supports transformation between schema and Java objects and between XML instance documents and Java object instances. JAXB consists of a runtime application programming interface (API) and accompanying tools that simplify access to XML documents. JAXB also helps to build XML documents that both conform and validate to the XML schema. The application server supports the W3C XML Schema as defined in the XML Schema 1.0 Recommendation (XSD Part 1 and 2).

JAXB-annotated classes and artifacts contain all the information needed by the JAXB runtime API to process XML instance documents. The JAXB runtime API supports marshaling JAXB objects to XML and unmarshaling the XML document back to JAXB class instances. Optionally, you can use JAXB to provide XML validation to enforce both incoming and outgoing XML documents to conform to the XML constraints defined within the XML schema.

JAXB is the default data binding technology that the Java API for XML web services (JAX-WS) tooling uses and is the default implementation within this product. You can develop JAXB objects for use within JAX-WS applications.

WebSphere® Application Server Version 7.0 supports the JAXB 2.1 specification. JAX-WS 2.1 requires JAXB 2.1 for data binding. JAXB 2.1 provides enhancements such as improved compilation support and support for the @XMLSeeAlso annotation, and full schema 1.0 support. With JAXB 2.1, you can configure the xjc schema compiler so that it does not automatically generate new classes for a particular schema. Similarly, you can configure the schemagen schema generator to not automatically generate a new schema. This enhancement is useful when you are using a common schema and you do not want a new schema generated. JAXB 2.1 also introduces the @XMLSeeAlso annotation that enables JAXB to bind additional Java classes that it might not otherwise know about when binding a Java class with this annotation. This annotation enables JAXB to know about all classes that are potentially involved in marshalling or unmarshalling as it is not always possible or practical to list all of the subclasses of a given Java class. JAX-WS 2.1 also supports the use of the @XMLSeeAlso annotation on a service endpoint interface (SEI) or on a service implementation bean to ensure all of the classes referenced by the annotation are passed to JAXB for processing.

For additional information refer to the official JSR-222 specification: [JSR-222: Java Architecture for XML Binding \(JAXB\) 2.0](#)

Parent topic: [Developing web service applications](#)

Related concepts:

[Java API for XML based web services](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Web services runtime environments](#)

[SOAP](#)

[SOAP MTOM](#)

[🔗 JSR 109 - Implementing enterprise web services](#)

[Annotations for creating web services overview](#)

Related tasks:

[Creating a JAX-WS-enabled WebSphere server](#)

[Creating a JAX-WS enabled web project](#)

[Creating a Java bean skeleton from a WSDL document using the WebSphere JAX-WS runtime environment](#)

Creating a web service from a Java bean using the IBM WebSphere JAX-WS runtime environment

Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment

Setting web services preferences

Developing JAX-RS applications

You can develop Java™ API for RESTful web services (JAX-RS) applications so that you can create Representational State Transfer (REST) services quickly.

- **Java API for RESTful web services**

Java API for RESTful web services (JAX-RS), also known as JSR-339, is a programming model that you can use to create Representational State Transfer (REST) services quickly.

- **Generating a JAX-RS resource from a WADL file**

You can generate a Java API for RESTful Web Services (JAX-RS) resource from a Web Application Description Language (WADL) file in a web service application.

- **wadl2java command**

Use the **wadl2java** command to generate Java API for RESTful Web Services (JAX-RS) resources from a Web Application Description Language (WADL) file in a web service application.

- **Generating sample client code for a JAX-RS 2.0 resource**

You can generate sample client code for a Java API for RESTful Web Services (JAX-RS) resource from a REST project address.

Parent topic: [Developing web service applications](#)

Java API for RESTful web services

Java™ API for RESTful web services (JAX-RS), also known as JSR-339, is a programming model that you can use to create Representational State Transfer (REST) services quickly.

Important: Applicable editions: Liberty profile, full profile

To develop JAX-RS applications, you must do these tasks:

- Install at least one of the following versions of WebSphere Application Server.
 - WebSphere® Application Server V7.0 with the Feature Pack for Web 2.0 and Mobile.
 - WebSphere Application Server V8.0 or later
 - WebSphere Application Server Liberty profile V8.5.5 or later
- Create an application in a project using the JAX-RS template. This template enables the JAX-RS facet, and required components of the Web 2.0 facet. This adds the library, servlet information, and support for JAX-RS annotations processing and JAX-RS quick fixes to your project.

To see how to create a JAX-RS application by using the IBM® JAX-RS implementation, see [Tutorial: Creating a JAX-RS web service](#).

For more information, see the official specification: [JSR 339: JAX-RS: The Java API for RESTful web services](#)

Parent topic: [Developing JAX-RS applications](#)

8.5.5.4 Generating a JAX-RS resource from a WADL file

You can generate a Java™ API for RESTful Web Services (JAX-RS) resource from a Web Application Description Language (WADL) file in a web service application.

Before you begin

Important: Applicable edition: Liberty profile

Complete the following setup for WebSphere® Application Server Developer Tools for Eclipse:

- Add the Liberty run time to the preferences.
- Create a Liberty server in the Servers view.
- Create a web project that targets the Liberty runtime environment.
- Import a WADL file into the workspace.

Procedure

1. In the Enterprise Explorer view, locate the WADL file.
2. Right-click the file and select **Web Services > Generate the JAX-RS Service from WADL**. The JAX-RS Service Generation from WADL dialog is displayed.
 - The **Source Folder** field specifies the destination folder of the generated client.
 - The **Name** field specifies the Java class name of the client.
 - The **Package** field specifies the Java package name of the client.
3. Accept the defaults and click **Finish**, or make modifications as appropriate.

Results

You created a JAX-RS resource. The generated code is in the destination folder.

What to do next

You can deploy the web application and test out the service. You can also [generate sample client code from your JAX-RS resource](#).

Parent topic: [Developing JAX-RS applications](#)

wadl2java command

Use the **wadl2java** command to generate Java™ API for RESTful Web Services (JAX-RS) resources from a Web Application Description Language (WADL) file in a web service application.

Important: Applicable edition: Liberty profile

The command has the same function as the Generate JAX-RS Service from WADL dialog.

Syntax

```
wadl2java.bat -p <package_name> -resource <name_of_generated_code> -d <output_directory> -impl
```

Options

- **-p**
 - Specifies the Java package of the generated code.
- **-resource**
 - Specifies the simple class name to use for the generated code.
- **-d**
 - Specifies the output directory for the generated JAX-RS resource.
- **-impl**
 - Specifies that a dummy service implementation is generated. This option is implicitly turned on in the wizard.

Parent topic: [Developing JAX-RS applications](#)

8.5.5.4 Generating sample client code for a JAX-RS 2.0 resource

You can generate sample client code for a Java™ API for RESTful Web Services (JAX-RS) resource from a REST project address.

Before you begin

Important: Applicable edition: Liberty profile

Complete the following setup for WebSphere® Application Server Developer Tools for Eclipse:

- Add the Liberty run time to the preferences.
- Create a Liberty server in the Servers view.
- Create a web project with one or more REST services that are defined in the project. Otherwise, if you do not define any services, the REST node has no children.

Procedure

1. Expand **your_REST_project** > **Services** > **REST**.
2. Right-click the address or one of its children and select **Generate** > **JAX-RS Client**. The JAX-RS 2.0 Client dialog is displayed.
 - A. In the **Source Folder** field, accept the default or specify the destination folder of the generated client.
 - B. In the **Name** field, accept the default or specify the Java class name of the client.
 - C. In the **Package** field, accept the default or specify the Java package name of the client.
 - D. Select the resources that you want to use to generate the client code.
 - E. Generate a new filter class.
 1. Select **Generate JAX-RS Client Filter**.
 2. Accept the default or specify the name of the filter class.This option creates a bare-bones class that implements the `ClientRequestFilter` and `ClientResponseFilter` interfaces from the JAX-RS 2.0 `javax.ws.rs.client` package. You can complete the implementation details for the interface methods after you generate the new filter class.
3. Optional: **8.5.5.6** To generate JAXB classes and the client into the same project, select **Use XML Schema for JAXB Generation**. The resource method of the JAX-RS service might have JAXB classes as method arguments or return types. If you do not have the JAXB source, but you do have the XML Schema, you can generate the JAXB classes so that your client code can access them.
 - A. Click **Next**. The JAX-RS Client with JAXB dialog is displayed.
 - B. Optional: In the **JAXB Target Java Package** field, provide the Java Package that you want to use for the JAXB generated classes. This value is passed to the binding compiler, or the Java Architecture for XML Binding Compiler (XJC) tool, with the `-p` option.
 - C. Click **Browse...** to browse to the XSD file from which you want to generate the JAXB class. You can also enter the XSD file into the **XML Schema**. This value is passed to the binding compiler, or the XJC tool, with the `-b` option.
 - D. Optional: To provide user-defined JAXB binding files, select **Specify JAXB binding files**. If you select this option, you can click **Add** to add custom binding declaration files. You can also remove files by highlighting them, and then selecting **Remove**.
 - E. Optional: To allow vendor extensions, select **Allow vendor extensions**. This selection might be necessary for processing some XML Schema documents. This value is passed to the binding compiler, or the XJC tool, with the `-extension` option.

4. Click **Finish**.

Results

You generated sample client code for a JAX-RS resource. **8.5.5.6** If you completed the optional step to generate JAXB classes, you also generated JAXB classes into the same project as your JAX-RS client.

You also generated a new Java class file by using the JAX-RS 2.0 Client API that can be used to access the JAX-RS service. Every annotated method of the JAX-RS service has a corresponding JAX-RS client method, which is generated from annotations on the JAX-RS service methods. The JAX-RS service methods include `@PathParam`; `@QueryParam`; `@Consumes`; `@Produces`; `@Path`; and resource methods such as `@GET`, `@PUT`, or `@POST`.

Restriction: All classes that are affected by the JAX-RS client generation algorithm must adhere to the following restrictions:

- These classes must be a string, a primitive, a JAX-RS API class, or a user class that is annotated by Java Architecture for XML Binding (JAXB). These classes cannot be binary.
- All classes must be in source form in order for the algorithm to locate them. If the classes are not in the same project as the JAX-RS service, the JAX-RS client cannot be generated.

What to do next

Add code to the sample so that the client code works and meets your JAX-RS 2.0 client code needs. To test the sample, add the client code to a JSP or a servlet. Your client code must be in a servlet container because there is no support for stand-alone Java applications. After you have working JAX-RS 2.0 client code, use the client code to start the JAX-RS 2.0 service.

Parent topic: [Developing JAX-RS applications](#)

Related information:

[Deploying JAX-RS applications to the Liberty profile](#)

Tools for web services development

Tools are provided to assist with the following aspects of web services development:

- **Create or Transform.** Create bottom-up web services from existing artifacts, such as Java™ beans and enterprise beans. Create top-down web services from WSDL discovered from others or created using the WSDL Editor.
- **Build.** Wrap existing artifacts as SOAP accessible services and describe them in WSDL. The web services wizards assist you in generating a Java client proxy to web services described in WSDL and in generating Java bean skeletons from WSDL.
- **Deploy.** Deploy web services into a variety of test environments.
- **Test.** Test web services running locally or remotely in order to get instant feedback.
- **Develop.** Generate sample applications to assist you in creating your own web service client application.

Services view

The Services view within the Java EE and web perspectives allows web services developers to quickly access a variety of tools which simplify web services development. Although these tasks can also be performed in the Enterprise Explorer view, the Services view only shows services and clients making it easier to find what you are looking for, and context menus specific to web services development.

The JAX-WS tools you can launch from the Services view include the following:

- WSDL interface editor

- You can launch the WSDL file for your web service or client in the WSDL editor in order to view or edit it. This is available for both static WSDL files in your workspace, and dynamic WSDL files generated by the runtime. Note that dynamic WSDL files cannot be edited.

- Deployment descriptor editors

- If your web service or client has deployment descriptors available, you can launch the Deployment Descriptor Editor to edit the `webservices.xml` file.

- Generate deployment descriptors

- If you have not previously generated a deployment descriptor, you can do so from this view. Although deployment descriptors are not required for JAX-WS web services because the runtime can generate this information on the fly, by generating deployment descriptors into your workspace you can customize the deployment settings.

- Generate a web service or client

- Depending on the object selected, you may be able to generate a top-down or bottom-up web service, or a web service client.

- [Full profile] Manage the policy sets for the web service or client

- You can apply and edit WebSphere® policy sets which regulate the qualities of service for your web services and clients.

- Test with the web Services Explorer

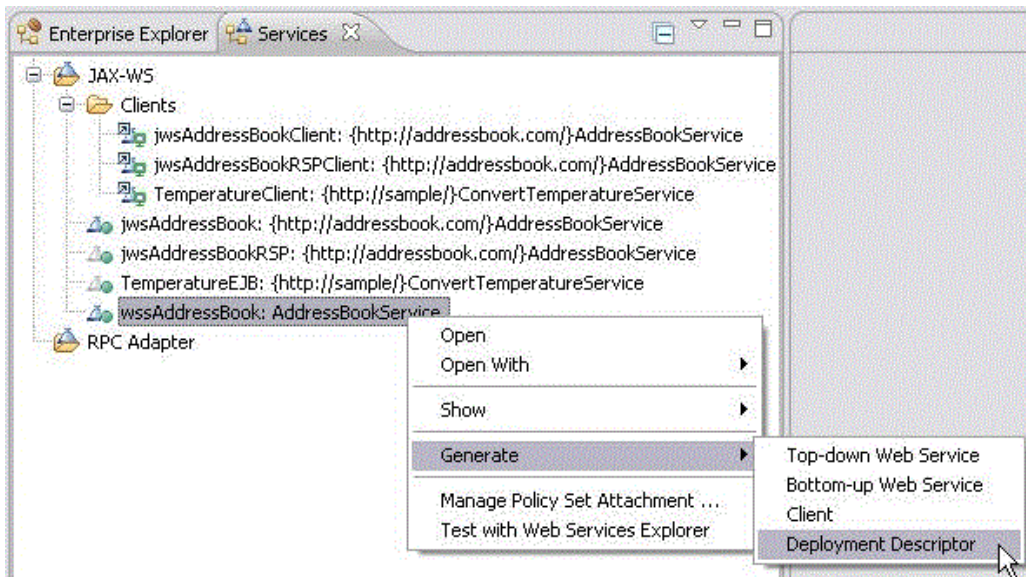
- You can test the function of your web service based on either a static or dynamic WSDL file using the web Services Explorer.

- Configure JAX-WS web service handlers

- This launches the JAX-WS Service Handlers Configuration wizard, which enabled you to add or edit handlers for your JAX-WS web service.

- [Full profile] Create router modules

- This option is available for EJB web services. The Create Router Modules wizard enables a set of web services within an enterprise application archive (EAR) file. For each web service-enabled EJB JAR in the EAR file, it adds an HTTP router, a JMS router, or both to the EAR. Each router module provides a web service endpoint for a particular transport.



- **Web services runtime environments**

Three runtime environments are available in which web services can run. Each environment supports different servers and standards.

Parent topic: [Developing web service applications](#)

Web services runtime environments

Three runtime environments are available in which web services can run. Each environment supports different servers and standards.

The following runtime environments are supported for web service generation:

- [IBM® WebSphere® JAX-WS runtime environment](#)
- [Apache Axis 1.4 runtime environment](#)

The IBM WebSphere JAX-WS runtime environment is the strategic web services runtime supported by IBM. New applications should use JAX-WS whenever possible. The Apache Axis 1.4 runtime is an open source runtime supported by the tools, however it is not recommended for use in a production environment.

Note that not all runtime environments support all methods of web service generation. For more information on what kinds of resources with which you can create a service, refer to the topics describing each runtime environment.

The following runtime/server configurations are supported:

- IBM WebSphere JAX-WS

- WebSphere Application Server v6.1.1.3 or later with the Feature Pack for web services installed, or WebSphere Application Server v7.0, v8.0 or v8.5 and up.

- Apache Axis 1.4

- Supported on all servers supporting servlet API 2.3 or higher. This includes but is not limited to: Apache Tomcat v4.x, v5.x, v6.0. WebSphere Application Server v6.1, v7.0.

- [IBM WebSphere JAX-WS runtime environment](#)

Java™ API for XML-Based Web Services (JAX-WS) is the next generation web services programming model. Using JAX-WS, development of web services and clients is simplified with greater platform independence for Java applications by the use of dynamic proxies and Java annotations.

Parent topic: [Tools for web services development](#)

Related concepts:

[Java API for XML based web services](#)

[JAXB](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Annotations for creating web services overview](#)

Related tasks:

[Creating a JAX-WS-enabled WebSphere server](#)

[Creating a JAX-WS enabled web project](#)

[Creating a Java bean skeleton from a WSDL document using the WebSphere JAX-WS runtime environment](#)

[Creating a web service from a Java bean using the IBM WebSphere JAX-WS runtime environment](#)

[Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment](#)

[Setting web services preferences](#)

IBM WebSphere JAX-WS runtime environment

Java™ API for XML-Based Web Services (JAX-WS) is the next generation web services programming model. Using JAX-WS, development of web services and clients is simplified with greater platform independence for Java applications by the use of dynamic proxies and Java annotations.

Enabling the WebSphere JAX-WS runtime environment

You can create top-down or bottom-up Java web services and clients. EJB web service generation using the JAX-WS runtime environment is only supported by WebSphere® Application Server V7.0 and later, and WebSphere Application Server Liberty Profile V8.5.5 and later.

The JAX-WS run time is enabled by default in WebSphere Application Server V7.0 and later. When you create web projects, so long as you select the default configuration for your version of WebSphere Application Server the correct facets are selected.

JAX-WS web service artifacts

A JAX-WS web service is contained within a web archive (WAR) file or a WAR module within an enterprise archive (EAR) file. A JAX-WS enabled WAR file contains:

- A WEB-INF/web.xml file.

- The `web.xml` does not contain servlet or servlet-mapping elements. The WebSphere Application Server run time defines them dynamically as the module is loaded.

- Annotated classes that implement the web services, which are contained in the application module.

- Annotated classes must contain, at a minimum, a web service implementation class that includes the `@javax.jws.WebService` annotation. The definition and specification of the web services-related annotations are provided by the JAX-WS and JSR-181 specifications. The web service implementation classes can exist within the WEB-INF/classes or directory within a Java archive (JAR) file that is contained in the WEB-INF/lib directory of the WAR file.

- Web Services Description Language (WSDL) documents that describe the web services that are contained in the application module [optional].

- You can optionally include WSDL documents in the JAX-WS application packaging. If the WSDL document for a particular web service is omitted, then the WebSphere Application Server runtime environment constructs the WSDL definition dynamically from the annotations that are contained in the web service implementation classes. You must include the `@javax.jws.WebService`, and optionally the `@WebMethod`, `@WebParam`, `@WebResult`, and `@SOAPBinding` annotations if the WSDL document is omitted.

Parent topic: [Web services runtime environments](#)

Parent topic: [Creating web services and clients by using the web service wizards](#)

Related concepts:

[Java API for XML based web services](#)

[JAXB](#)

[Web services runtime environments](#)

[Annotations for creating web services overview](#)

Related tasks:

[Creating a JAX-WS-enabled WebSphere server](#)

Creating a JAX-WS enabled web project

Creating a Java bean skeleton from a WSDL document using the WebSphere JAX-WS runtime environment

Creating a web service from a Java bean using the IBM WebSphere JAX-WS runtime environment

Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment

Setting web services preferences

Configuring a workspace for web services development

Although you can begin web services development immediately upon creating a workspace, you might find it convenient to configure your workspace to optimize your development experience.

About this task

After you set up your workspace, you can begin to develop web services by creating or importing the resources that your web service or web service client will use.

- [Setting web services preferences](#)

Before you begin developing web services or clients, you can optimize the workbench for web services development by setting a variety of preferences.

- [Setting the level of WS-I compliance](#)

The web services WS-I validation tools support the level of WS-I compliance outlined in the WS-I Basic Profile 1.1, 1.2, 2.0, the WS-I Simple SOAP Binding Profile 1.0 (WS-I SSBP), the WS-I Attachments Profile 1.0 (WS-I AP), and the WS-I Basic Security Profile 1.0 (WS-I BSP). You can choose to make your web service compliant or non-compliant, depending on your needs. For example, encoded style (RPC/encoded), SOAP over JMS protocols are not WS-I compliant.

- [Creating a JAX-WS-enabled WebSphere server](#)

JAX-WS web services can only be targeted to servers that support the JAX-WS runtime environment, such as IBM® WebSphere Application Server Liberty Profile or WebSphere Application Server V7.0 or later.

- [Creating a WebSphere Application Server and Web project](#)

If you plan to create a web service that uses WebSphere Application Server as its server, the required version of WebSphere Application Server must be installed and a server created before you begin creating your web service.

- [Creating a Liberty profile server and Web project](#)

If you plan to create a web service that uses WebSphere Application Server Liberty Profile V8.5.5, the server must be installed and created before you begin creating your web service.

- [Creating a JMS server](#)

In order to create a web service that uses JMS transports, you need to first create and configure a server that can support JMS.

- [Configuring the IBM JRE to talk to a secured WebSphere Application Server](#)

Use these steps if you want to use the web services wizard to retrieve an HTTPS WSDL or if you want to use the Web Services Explorer against a secured WebSphere Application Server. If you encounter an error similar to `Error opening socket: javax.net.ssl.SSLHandshakeException: unknown certificate` this task resolves the issue. This error occurs because WebSphere Application Server uses a security certificate for negotiating secured connections that other JRE-based applications do not normally share.

- [Problems working with a secured server using SSL connections](#)

How to work around the `SSLConnectionFactory` and `SSLHandshakeException` error messages when you are trying to communicate to a secured server by using a Secure Sockets Layer (SSL) connection within the workbench.

Parent topic: [Developing web service applications](#)

Related tasks:

[Setting web services preferences](#)

[Setting the level of WS-I compliance](#)

[Creating a WebSphere Application Server and Web project](#)

Creating a Liberty profile server and Web project

Configuring the IBM JRE to talk to a secured WebSphere Application Server

Setting web services preferences

Before you begin developing web services or clients, you can optimize the workbench for web services development by setting a variety of preferences.

About this task

To set any of the web services preferences, follow these steps:

Procedure

1. Click **Window > Preferences** to open the Preferences notebook.
2. Expand **Web services** and click the preference category that you want to set.
3. Select the check boxes that you want to set as the default when creating your web service.
4. Click **OK** to apply the changes and close the Preferences notebook.

What to do next

For information on individual preferences, refer to the following:

- [WebSphere® web services preferences](#)
- [Web services wizards basic preferences](#)
- [WSDL file preferences](#)
- [Service policies preferences](#)
- [Axis emitter preferences](#)

- **WebSphere Web services preferences**

The WebSphere Web services preferences page allows you to set the following preferences used when creating or consuming WebSphere Web services.

- **Service policies web services preferences**

The **Preferences > General > Service policies** page allows you to set WS-I, and policy set, and binding defaults.

- **Web Service Explorer preferences**

Parent topic: [Configuring a workspace for web services development](#)

Related concepts:

[Java API for XML based web services](#)

[JAXB](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Web services runtime environments](#)

WebSphere Web services preferences

The WebSphere® Web services preferences page allows you to set the following preferences used when creating or consuming WebSphere Web services.

Security

- Show only FIPS compliant algorithms:

- Select this if you only want the FIPS compliant algorithms to be shown in the Data Encryption method algorithm and Key Encryption method algorithm drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the Use the Federal Information Processing Standard (FIPS) option in the Global security panel of the WebSphere administrative console.

- Show "*" instead of letters for passwords

- By default the workbench will display an asterisk "*" rather than letters for passwords.

JAX-WS Code generation

- Top Down

- **Copy WSDL into project:** Select this to copy the WSDL file from which the Web service is being created into the Web service project. This is a convenient option if you plan to create the client at a later time or publish the WSDL for other users.
- **Enable wrapper style:** For WSDL documents that implement a document/literal wrapped pattern, a root element is declared in the XML schema and is used as an operation wrapper for a message flow. Separate wrapper element definitions exist for both the request and the response. More simply, the element whose name is the same as the operation (the wrapper element) is broken apart so that each of its content elements becomes a parameter of the generated Java™ method signature.
- **Generate serializable JAXB classes:** In WebSphere Application Server v7.0 and v8.0 when you enable the Java 6 facet, you can choose to generate JAXB classes which implement java.io.Serializable. Classes that do not implement this interface will not have any of their state serialized or deserialized.
- **Enable MTOM support:** If you select this check box the SOAP Message Transmission Optimization Mechanism will be enabled.
- **Generate schema library project from XSD file:** Selecting this will run the JAX-WS Schema to Java compiler to generate schema code into a schema library.
- **Generate Web service deployment descriptor:** For JAX-WS Web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer required. Selecting this checkbox will generate them.
- **Version of JAX-WS code to be generated:** When deploying to WebSphere Application Server v7.0 or later, you can generate JAX-WS 2.1 compliant code, and when using WebSphere Application Server v8.0 you can generate JAX-WS 2.2 compliant code. Previous versions of the server only support JAX-WS 2.0.

- Bottom Up

- **Enable SOAP 1.2 support:** You can choose between SOAP 1.1 and SOAP 1.2 bindings.
- **Enable MTOM support:** If you select this check box the SOAP Message Transmission Optimization Mechanism will be enabled.
- **Java to WSDL mapping style:** This specifies the style of Java to WSDL mapping. Style defines encoding style for messages sent to and from the Web service. Parameter style determines whether the method's parameters represent the entire message body or whether parameters are elements wrapped inside a top-level element named after the operation. The combinations are RPC, Document/Wrapped, or Document/Bare.
- **Generate WSDL into the project:** Select this to generate a WSDL file into the Web service project. This is a convenient option if you plan to create the client at a later time or publish the WSDL for other users.

- **Generate Web service deployment descriptor:** For JAX-WS Web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer required. Selecting this checkbox will generate them.

- Client

- **Enable asynchronous client support:** If you select to enable an asynchronous client, for each method in the Web service two additional methods will be created. These are polling and callback methods which allow the client to function asynchronously.

- **Generate serializable JAXB classes:** In WebSphere Application Server v7.0 and v8.0 when you enable the Java 6 facet, you can choose to generate JAXB classes which implement `java.io.Serializable`. Classes that do not implement this interface will not have any of their state serialized or deserialized.

- **Generate portable client:** Selecting this checkbox would allow you to move your Web service client code from one machine to another or from one instance of WebSphere Application Server to another. If this option is selected, the WSDL document and all the XML Schema and other WSDL documents that it depends upon will be copied into the client project under `WEB-INF/wsdl` and a `file:relativeURL` pointing to this copy will then be injected into the JAX-WS Service class's static initialization block.

- **Generate Web service deployment descriptor:** For JAX-WS Web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer required. Selecting this checkbox will generate them.

- **Generate ibm-webservicesclient-bnd.xml template for overriding the service's endpoint URL:** When enabled, results in the generation of a WebSphere extended deployment descriptor which overrides the endpoint URL used by clients when invoking a Web service. This only takes effect when the client is using JSR-109 (that is, has a client service-ref deployment descriptor). Otherwise, the client runs in unmanaged form and ignores this setting. If you generate deployment descriptors and try to use a TCP/IP monitor, the endpoint is fixed, and the TCP/IP monitor will not see any traffic.

- **Enable MTOM support:** If you select this check box the SOAP Message Transmission Optimization Mechanism will be enabled.

- **Version of JAX-WS code to be generated:** When deploying to WebSphere Application Server v7.0 or later, you can generate JAX-WS 2.1 compliant code. Previous versions of the server only support JAX-WS 2.0.

Parent topic: [Setting web services preferences](#)

Service policies web services preferences

The **Preferences > General > Service policies** page allows you to set WS-I, and policy set, and binding defaults.

Profile compliance

This section of the service policies page can be used to set the level of WS-I compliance. For more information on WS-I, refer to their Web site: <http://www.ws-i.org/>.

- WS-I AP 1.0 (WS-I Attachments Profile 1.0)

- Supports interoperable SOAP messages with attachments-based web services.

- WS-I BP 1.1 + SSBP 1.0 (WS-I Basic Profile and WS-I Simple SOAP Binding Profile)

- This includes the basic profile and requirements related to the serialization of an envelope and its representation in a SOAP message.

- WS-I BP 1.2 (WS-I Basic Profile)

- The WS-I Basic Profile 1.2 builds on Basic Profile 1.1 by incorporating Basic Profile 1.1 errata and requirements from Simple SOAP Binding Profile 1.0, and adding support for WS-Addressing and MTOM.

- WS-I BP 2.0 (WS-I Basic Profile)

- The WS-I Basic Profile 2.0 consists of a set of non-proprietary web services specifications, along with clarifications, refinements, interpretations and amplifications of those specifications which promote interoperability.

- WS-I BSP 1.0 (WS-I Basic Security Profile)

- The Basic Security Profile 1.0 provides guidance on the use of WS-Security and the REL, Kerberos, SAML, UserName and X.509 security token formats.

For each profile you can select from three levels of compliance with WS-I specifications:

- Require WS-I compliance - this level prevents you from creating a non-compliant web service.
- Suggest WS-I compliance - this level allows you to create a non-compliant web service, but provides a visible warning stating how the service is non-compliant.
- Ignore WS-I compliance - this level allows you to create a non-compliant web service and does not notify you of non-compliance.

WebSphere general bindings

Important: Applicable edition: Full profile

Policy set bindings contain platform specific information, like keystore, authentication information or persistent information, required by a policy set attachment. General bindings, new in WebSphere® Application Server v7.0, can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Although general bindings are highly reusable, they do not provide configuration for advanced policy requirements, such as multiple signatures.

Bindings can be created in the WebSphere Application Server administrative console and then optionally imported into the development workspace using **Import > Web services > WebSphere named bindings**. Once service and client bindings have been created, you can apply them to the entire workspace or a single project as the default binding using the Service policies preference page.

The sample general bindings shipped with the product are the following:

- **Client and provider sample:** For more information, refer to [General sample bindings for JAX-WS applications](#)
- **Client and provider V2 sample:** For more information, refer to [General sample bindings for JAX-WS applications](#)
- **SAML bearer sample:** For more information, refer to [Configuring client and provider bindings for the SAML bearer token](#)
- **SAML HoK symmetric sample:** For more information, refer to [Configuring client and provider bindings for the SAML holder-of-key symmetric key token](#)

Important:

- The general bindings that are shipped with the product are provider and client sample bindings. These bindings are

initially set as the cell default bindings. Do not use these bindings in their current state in a production environment. To use the sample bindings, modify them to meet your security needs in a production environment. Alternatively, create a copy of the bindings and then modify the copy.

- You cannot assign a binding to a service provider resource that does not have a policy set or has an inherited attachment. To assign a binding to such a service provider resource, you must first attach a policy set to the resource. Also, you cannot assign a binding to a service client resource that does not have an effective policy configuration or has an inherited policy attachment. To assign a binding to such a service client resource, you must first attach a policy set or specify the use of the provider policy.

For more information on general bindings and how to create them, refer to the WebSphere Application Server v7.0 documentation topic: [Defining and managing policy set bindings](#).

WebSphere Application Server policy sets

Important: Applicable edition: Full profile

You can use the preferences page to select the default policy set for web services and clients. Several policy sets are included with the workbench by default when you choose to install a runtime or stub, while additional policy sets can be imported.

The following policy sets are included with the workbench:

- WebSphere Application Server Policy Sets

- **Kerberos v5 HTTPS default.** This policy set provides message authentication with a Kerberos Version 5 token. Message integrity and confidentiality are provided by Secure Sockets Layer (SSL) transport security. This policy set follows the OASIS Kerberos Token Profile V1.1 and WS-Security specifications. When you use this policy set, configure the basic authentication data and custom properties such as the `com.ibm.wsspi.wssecurity.krbtoken.targetServiceName` and `com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost` custom properties in the client bindings. For more information, see the Authentication generator or consumer token settings and Protection token settings (generator or consumer) topics in the WebSphere Application Server Information Center.
- **LTPA WSSecurity default.** This policy set provides the following features:
 - Message integrity by digital signature (using RSA public-key cryptography) to sign the body, timestamp, and WS-Addressing headers using WS-Security specifications
 - Message confidentiality by encryption (using RSA public-key cryptography) to encrypt the body, signature and signature confirmation elements using WS-Security specifications
 - A Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service
- **SSL WSTransaction.** This policy set enables WS-Transaction, which provides the ability to coordinate distributed transactional work atomically, interoperably and securely using the WS-AtomicTransaction specification and SSL Transport security.
- **Username SecureConversation.** This policy set provides the following features:
 - Message integrity by digital signature that includes signing the body, timestamp, and WS-Addressing headers using WS-SecureConversation and WS-Security specifications
 - Message confidentiality by encryption that includes encrypting the body, signature and signature confirmation elements, using WS-SecureConversation and WS-Security specifications
 - A username token included in the request message to authenticate the client to the service. The username token is encrypted in the request
- **Username WSSecurity default.** This policy set provides the following features:
 - Message integrity by digital signature (using RSA public-key cryptography) to sign the body, timestamp, and WS-Addressing headers using WS-Security specifications

- Message confidentiality by encryption (using RSA public-key cryptography) to encrypt the body, signature and signature confirmation elements using WS-Security specifications
- A username token included in the request message to authenticate the client to the service. The username token is encrypted in the request
- **WS-I RSP** This policy set enables WS-ReliableMessaging Version 1.1 and uses the minimum quality of service, unmanaged non-persistent, which provides the ability to deliver a message reliably to its intended receiver. This policy set only works in a single server environment and does not work in a clustered environment. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages. In-order delivery is set to "false", so messages are not necessarily delivered in the order in which they were sent. Message integrity is provided by digitally signing the body, the time stamp, and the WS-Addressing headers. Message confidentiality is provided by encrypting the body and the signature. This policy set follows the WS-SecureConversation and WS-Security specifications.
- **WSAddressing default.** The WSAddressing default policy set provides a transport-neutral way to uniformly address web services and messages. This policy set enables WS-Addressing support, which uses endpoint references and message addressing properties to facilitate the addressing of web services in a standard and interoperable way.
- **WSHTTPS default** This policy set provides SSL transport security for the HTTP protocol with web services applications.
- **WSReliableMessaging persistent.** This policy set enables both WS-ReliableMessaging and WS-Addressing and uses the maximum quality of service, managed persistent. This quality of service supports asynchronous web service invocations and uses a service integration messaging engine and message store to manage the sequence state. Messages are processed within transactions, are persisted at the web service requester server and at the web service provider server, and are recoverable in the event of server failure. In-order delivery is set to "false", so messages are not necessarily delivered in the order in which they were sent. Because this policy set specifies managed persistent quality of service, you have to define bindings to the service integration bus and messaging engine that you want to use to manage the WS-ReliableMessaging state. You can attach and bind a WS-ReliableMessaging policy set to a web service application by using the WebSphere Application Server administrative console or the wsadmin tool.

- WebSphere Application Server System Policies

- **SystemWSSecurityDefault** This system policy set specifies the asymmetric algorithm and both the public and private keys to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using RSA encryption. Message confidentiality is provided by encrypting the body and signature using RSA encryption. This policy set follows the WS-Security specifications for the issue and renew trust operation requests. For more information on system policy sets, refer to: [System policy sets](#)

WebSphere Programming Models

- JAX-WS annotation scanning message severity

- Use this preference to set the level of error message generated for JAX-WS annotation scanning. The WebSphere JAX-WS runtime scans Web applications for JAX-WS annotations when the application is added to WebSphere Application server. For performance reasons WebSphere Application Server does not scan all projects for annotations by default. On WebSphere Application Server v7 and later, version 2.4 web modules are not scanned unless the `UseWSEFP61ScanPolicy` property is set to `true` in the MANIFEST.MF file. To inform you that WebSphere is not performing an annotation scan, the workbench adds an error marker. However if you are using a non-WebSphere implementation of the JAX-WS runtime, this error message is invalid, and may prevent you from publishing to WebSphere Application Server. Use this preference to change the level of error message severity for your workspace or project so that you can publish successfully.

- Provide JAX-WS 2.1.6 and later method exposure guidance

- JAX-WS 2.1.6 and later assistance can now be enabled when this is set to **true**. When this support is turned on, warnings will be issued for methods in Java™ classes that will be implicitly exposed as a result of the new specification and a quickfix will be available to hide these methods and preserve the web service interface from prior JAX-WS standards. For more information on JAX-WS 2.1.6 refer to: [Java API for XML based web services](#)

Parent topic: [Setting web services preferences](#)

Web Service Explorer preferences

Changing the timeout settings for the Web Service Explorer

Although the timeout setting does not need to be altered in most situations, you can change this setting at the product or workspace level. This preference represents the session timeout of the Web Service explorer in minutes. Entering a negative value will disable timeouts completely. If the preference is not set the default is 30 minutes.

To change the setting at the product level:

1. Open your product's `plugin_customization.ini` file.
2. Add the following line:`org.eclipse.wst.ws.explorer/sessionTimeout=<timeout value>`

To change the setting at the workspace level:

1. Open the following file, or create it if it does not exist:

```
<workspace_name>\.metadata\.plugins\org.eclipse.core.runtime\.settings\org.eclipse.wst.ws.explorer.prefs
```

2. Add or update the following line: `sessionTimeout=<timeout value>`

Note: The default timeout setting is designed to prevent a memory leak. If you disable the timeout setting completely, this safeguard will no longer be in place.

Parent topic: [Setting web services preferences](#)

Setting the level of WS-I compliance

The web services WS-I validation tools support the level of WS-I compliance outlined in the WS-I Basic Profile 1.1, 1.2, 2.0, the WS-I Simple SOAP Binding Profile 1.0 (WS-I SSBP), the WS-I Attachments Profile 1.0 (WS-I AP), and the WS-I Basic Security Profile 1.0 (WS-I BSP). You can choose to make your web service compliant or non-compliant, depending on your needs. For example, encoded style (RPC/encoded), SOAP over JMS protocols are not WS-I compliant.

About this task

For more information on WS-I, refer to their web site: <http://www.ws-i.org/>. This site contains resources such as an overview of web services interoperability, usage scenarios, and specifications.

WS-I Basic Profile is a outline of requirements to which WSDL and web service protocol (SOAP/HTTP) traffic must comply in order to claim WS-I conformance. The web services WS-I validation tools currently support the following:

- WS-I AP 1.0 (WS-I Attachments Profile 1.0)

- Supports interoperable SOAP messages with attachments-based web services.

- WS-I BP 1.1 + SSBP 1.0 (WS-I Basic Profile and WS-I Simple SOAP Binding Profile)

- This includes the basic profile and requirements related to the serialization of an envelope and its representation in a SOAP message.

- WS-I BP 1.2 (WS-I Basic Profile)

- The WS-I Basic Profile 1.2 builds on Basic Profile 1.1 by incorporating Basic Profile 1.1 errata and requirements from Simple SOAP Binding Profile 1.0, and adding support for WS-Addressing and MTOM.

- WS-I BP 2.0 (WS-I Basic Profile)

- The WS-I Basic Profile 2.0 consists of a set of non-proprietary web services specifications, along with clarifications, refinements, interpretations and amplifications of those specifications which promote interoperability.

- WS-I BSP 1.0 (WS-I Basic Security Profile)

- The Basic Security Profile 1.0 provides guidance on the use of WS-Security and the REL, Kerberos, SAML, UserName and X.509 security token formats.

To view the specifications, refer to the WS-I web site, and select the appropriate profile under **Deliverables**.

Depending on the type of web service being created, you may or may not want your web service to comply with the WS-I profiles. The default level of compliance is to generate a warning if a non WS-I SSBP compliant web service option is selected and to ignore any non WS-I AP compliant selections. You can set the level of WS-I compliance at the workspace or project level. The web services wizards, the WebSphere® run-time environments, the WSDL editor, and other web services tools provided support and encourage the development of WS-I compliance services.

For each profile you can select from three levels of compliance with WS-I specifications:

- Require WS-I compliance - this level prevents you from creating a non-compliant web service.
- Suggest WS-I compliance - this level allows you to create a non-compliant web service, but provides a visible warning stating how the service is non-compliant.
- Ignore WS-I compliance - this level allows you to create a non-compliant web service and does not notify you of non-compliance.

You can set the level of WS-I compliance at the workspace level, or at the project level.

Parent topic: [Configuring a workspace for web services development](#)

Set the level of WS-I compliance for the workspace

About this task

To set the level of WS-I compliance for the workspace:

Procedure

1. From the **Window** menu, select **Preferences**.
2. Select **Service Policies** from the component tree.
3. Expand **Profile Compliance** and select the profiles with which you want your web services to be compliant. Select the level of compliance you want from the drop-down lists.
4. Click **Apply**, then click **OK**.

Set the level of WS-I compliance for a project

About this task

To set the level of WS-I compliance for a project:

Procedure

1. In the Project Navigator, right-click and select **Properties**.
2. Select **Service Policies** from the component tree.
3. Select the check box to enable project-specific settings.
4. Expand **Profile Compliance** and select the profiles with which you want your web services to be compliant. Select the level of compliance you want from the drop-down lists.
5. Click **Apply**, then click **OK**.

What to do next

Note: Not all projects can have the level of WS-I compliance set, and therefore the Service Policies preferences page will not be visible for all projects. web projects and any project containing a WSDL file should have these settings available.

Creating a JAX-WS-enabled WebSphere server

JAX-WS web services can only be targeted to servers that support the JAX-WS runtime environment, such as IBM® WebSphere® Application Server Liberty Profile or WebSphere Application Server V7.0 or later.

Before you begin

If you plan to create a JAX-WS web service, before you generate deployment code for your web service you must create one of the following:

- IBM WebSphere Application Server Liberty Profile
- WebSphere Application Server Version 7.0 or later.

About this task

To create a WebSphere Application Server:

Procedure

1. From the File menu, select **New > Other > Server > Server > Next**.
2. Select the appropriate version of WebSphere Application Server as the server type, and click **Next**.
3. On the WebSphere server settings page, select the server profile that is associated with your WebSphere Application Server, and enter a server name.
4. Click **Finish**.

Results

To see the server you have created in the workspace, from the Window menu click **Show view > Other > Server > Servers > Ok** or open the Java™ EE perspective. By double-clicking the server you can view or modify many of the server's settings.

What to do next

If you are using a secured WebSphere Application Server, there are a couple of tasks you may need complete in order to use the web services tools:

- Before using the Web Services Explorer, generating sample JSPs, or deploying an Axis Web service, you must configure the workbench to communicate with a server using Secure Sockets Layer (SSL). Information about this task can be found here: [Problems working with a secured server using SSL connections](#).
- If you want to use the Web services wizard to retrieve an HTTPS WSDL or if you want to use the Web Services Explorer against a secured WebSphere Application Server, you must complete the steps in: [Configuring the IBM JRE to talk to a secured WebSphere Application Server](#)

Tip:[Full profile] By default the **Minimize application files copied to the server** option has been turned on in the server editor for performance reasons. Generally speaking this means you will not be able to see the application deployment descriptor files from the WebSphere Application Server administrative console. In the case of web services, both `Publish WSDL files` and `Provide HTTP endpoint URL information links` will be missing from the console because the `WSDL`, `ibm-web-bnd.xmi`, `ibm-web-ext.xmi`, and `web.xml` will not be in the `*.war` file copied to the server profile directory. This may cause problems if the WebSphere Application Server administrator does not have access to the files residing in the development workspace. To disable this option, double-click the server in the Servers view, and clear the **Minimize application files copied to the server** check box. This will have a small performance impact on the server but will make all the required files available to the administrator.

- **Creating a JAX-WS enabled web project**

Before creating a JAX-WS web service, you must create a web project with a Java Version 5 or later facet.

Parent topic: [Configuring a workspace for web services development](#)

Related concepts:

[Java API for XML based web services](#)

[JAXB](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Web services runtime environments](#)

Creating a JAX-WS enabled web project

Before creating a JAX-WS web service, you must create a web project with a Java™ Version 5 or later facet.

Before you begin

Before you create a web project you should create a WebSphere® server as described in: [Creating a JAX-WS-enabled WebSphere server](#)

About this task

To build a Web project that points to the WebSphere server that you have created:

Procedure

1. In the Java EE perspective, right-click your enterprise application project and select **New > Web Project** to open the web project wizard.
2. In the **Name** field, type a name for your new web project.
3. In the Project Templates section, select the type of web template you want to use: select **Simple** to create a simple web project.
4. In the Programming Model section, select the programming model that you want to use: select **Java EE**. Click **Next** to configure your new web project.
5. On the deployment page, from the list of available configuration options, click **Deployment** to open the Deployment configuration page.
 - You can change **Target runtime** by selecting another one from the drop-down box. Click **Change Features** to open the Project Facets window. If you are targeting WebSphere Application Server v7.0 or later, the default configuration is adequate for JAX-WS web service development.
 - Click **Add support for WebSphere bindings and extensions** or clear this field.
 - In the **Web module version** field, select the web module version that you want to use.
 - In the **EAR membership** field, click **Add project to an EAR**, if you want to include EAR membership; clear this field if you do not want to add the web project to an EAR file. The New project wizard generates the EAR.
 - In the **EAR project name** field, the name of your existent EAR file appears. You can click **Browse** to select a different EAR file.
 - Click **Finish**.

Note: The deployment option is not available if you selected the Client-side only programming model for your new web project.
6. From the list of available configuration options, click **Java** to open the Java configuration page.
 - In the **Source folders on build path** field, accept the default **src** directory, or click **Add Folder**, **Edit...** or **Remove** to specify a folder for your source files.
 - In the **Default output folder:** field, specify a folder for your output files or accept the default value (WebContent\WEB-INF\classes).
7. From the list of available configuration options, click **Web Module**. On the Web Module configuration page:
 - In the **Context root** field, type the name of your web project root, or accept the default (which is the name of your web project).
 - In the **Content directory** field, type the name of your content directory, or accept the default (WebContent).
 - Select **Generate web.xml deployment descriptor** if you want to create a deployment descriptor. You can also add a deployment descriptor to your web module later.
8. To associate the project with the server, right-click the WebSphere server that you created in the Servers view and click **Add and Remove Projects**. Select the service and client projects from the **Available projects** list and click **Add**.

Parent topic: [Creating a JAX-WS-enabled WebSphere server](#)

Related concepts:

[Java API for XML based web services](#)

[JAXB](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Web services runtime environments](#)

Creating a WebSphere Application Server and Web project

If you plan to create a web service that uses WebSphere® Application Server as its server, the required version of WebSphere Application Server must be installed and a server created before you begin creating your web service.

About this task

To create a WebSphere Application Server:

Procedure

1. From the File menu, select **New > Other > Server > Server > Next**.
2. Select WebSphere Application Server as the server type, electing the correct version of the server for your install, and click **Next**.
3. On the WebSphere Application Server settings page enter a server name.
4. Click **Finish**. To see the server you have created in the workspace, from the Window menu select **Show view > Other > Server > Servers > Ok**.

Example

To build a web project that points to the WebSphere Application Server that you have created:

1. Create the service Web project by selecting **New > Web > Web Project**.
2. Project Name: enter a project name
3. Target runtime: select the version of WebSphere Application Server that you have installed. If it is not listed, click **New** and browse to the location where it is installed.
4. Configuration: Accept the default configuration for your server, or click **Modify** to change the project facets.
5. Ensure that **Add project to an EAR** is selected. The EAR will be generated by the New project wizard.
6. Click **Finish**.
7. To associate the project with the server, right-click the WebSphere Application Server you created in the Servers view and select **Add and remove projects**. Select the service and client projects from the Available projects list and click **Add**.
8. Start the server by right-clicking it and selecting **Start**.

Parent topic: [Configuring a workspace for web services development](#)

Related tasks:

[Creating a WebSphere Application Server](#)

Creating a Liberty profile server and Web project

If you plan to create a web service that uses WebSphere® Application Server Liberty Profile V8.5.5, the server must be installed and created before you begin creating your web service.

About this task

To create a Liberty Profile server:

Procedure

1. From the File menu, select **New > Other > Server > Server > Next**.
2. Expand **IBM** and select WebSphere Application Server V8.5 Liberty Profile as the server type, electing the correct version of the server for your install, and click **Next**.
3. See the [Creating a Liberty profile server by using developer tools](#) task to complete the New Server wizard.
4. Click **Finish**. To see the server you have created in the workspace, from the Window menu select **Show view > Other > Server > Servers > Ok**.

Example

To build a web project that points to the Liberty Profile server that you have created:

1. Create the service Web project by selecting **New > Web > Web Project**.
2. Project Name: enter a project name
3. Target runtime: select **WebSphere Application Server V8.5 Liberty Profile** that you have installed. If it is not listed, click **New** and browse to the location where it is installed.
4. Configuration: Accept the default configuration for your server, or click **Modify** to change the project facets.
5. Ensure that **Add project to an EAR** is selected. The EAR will be generated by the New project wizard.
6. Click **Finish**.
7. To associate the project with the server, right-click the WebSphere Application Server V8.5 Liberty Profile you created in the Servers view and select **Add and remove projects**. Select the service and client projects from the Available projects list and click **Add**.
8. Start the server by right-clicking it and selecting **Start**.

Parent topic: [Configuring a workspace for web services development](#)

Creating a JMS server

In order to create a web service that uses JMS transports, you need to first create and configure a server that can support JMS.

Before you begin

Important: Applicable edition: Full profile

In order to create a server, you must have installed at least one of WebSphere® Application Server Version 7.0, or the WebSphere Application Server legacy test environments. JMS web services are only supported on WebSphere Application Server.

About this task

The following steps will guide you through basic service integration bus and JMS queue creation using the default messaging provider included with WebSphere Application server. Service integration technologies replaced the embedded messaging provider used in previous versions of the product. These instructions are not a definitive guide to how to use web services with service integration technologies, however they can be used as a guide when setting up topics and queues to handle JMS messages.

For additional information about the default messaging provider, including more detailed information on JMS and why and how to set up queues and topics for JMS, refer to the WebSphere Application Server information center and search for "messaging resources." For additional information on using service integration technologies with web services, refer to the WebSphere Application Server information center and search for Enabling web services through service integration technologies.

- [HTTP and JMS transport methods](#)

Web services created using the WebSphere runtime environments support a JMS transport layer in addition to the existing HTTP transport. This allows web service clients and servers to communicate using JMS queues and topics instead of HTTP connections. Both one-way and synchronous two-way requests are supported.

Parent topic: [Configuring a workspace for web services development](#)

Create a server and service integration bus for SOAP over JMS using WebSphere Application Server Procedure

1. From the **File** menu, select **New > Other > Server > Server > Next**.
2. Select one of the versions of WebSphere Application Server as the server type. Click **Next**.
3. If this runtime has not been created in your workspace, you will be prompted to select the installation directory for the server. Click **Next**.
4. Accept the default server port and name. Click **Next**.
5. Select the JMSEAR from the list of available projects and click **Add** to target it to the server. Click **Finish**.
6. Wait for the server to start. once it has started the console will display `Server server1 open for e-business;`
7. Launch the administrative console by right-clicking the server in your Servers view and selecting **Administration > Run administrative console**.
8. Expand **Servers > Server Types > WebSphere application servers** to ensure that the server you created is listed.

9. Expand **Service Integration > Buses** and click **New**. Enter a unique name in the Name field (for example `ws_test_bus`) and click **Next**, unselect **Bus security** and click **Next**, and then **Finish**.
 10. To associate the current server with the newly created integration bus, select the name of the bus you have just created. Under **Local Topology**, expand the name of your bus, expand **Bus members**, and then click **Add**. Select the server you want to associate the integration bus with and then click **Next**. Select **File store** as the message persistence state and click **Next**. You can accept the default message store properties for this tutorial and click **Next**. If you are creating a JMS bus for your own web service, select Help and search on File store settings for additional information about which settings are best for you. Click **Finish** to confirm.
 11. Create a physical queue for the request message:
 - A. Expand **Service Integration > Buses**. Select the bus created earlier (`ws_test_bus`).
 - B. Under **Destination resources** click **Destinations**.
 - C. On the destinations page click **New**.
 - D. Choose **Queue** as the destination type and click **Next**.
 - E. Enter an identifier such as `ws_test_queueJms`. Click **Next**.
 - F. Accept the default bus member. Click **Next**.
 - G. Click **Finish** to confirm your changes, and then save your changes.
 12. Assign JMS settings against the newly created queue:
 - A. Go to **Resources > JMS > JMS Providers**.
 - B. From the Scope drop-down list select the server as your scope, and from the provider list select **Default messaging provider**.
 - C. Under Additional Properties select **Queues**. Click **New**.
 - D. Enter a name (for example `ws_test_queueJms`) and JNDI name (for example `jms/ws_test_queue`). Select the bus (`ws_test_bus`) and Queue (`ws_test_queueJms`) you created earlier.
 - E. Click **OK** to save the changes.
 13. Create a queue connection factory for the input queue:
 - A. Select **Resources > JMS > Queue connection factories**.
 - B. From the Scope drop-down list select the server as your scope, and click **New**.
 - C. Select the default messaging provider and click **OK**.
 - D. Under General Properties enter a name (for example `webServicesInput_QCF`) and a JNDI name (for example `jms/ws_test_qcf`).
 - E. In the Connection pane select the bus created earlier (`ws_test_Bus`) as the bus name.
 - F. Click **OK** to save the changes.
 14. Create a queue connection factory for the reply queue:
 - A. Select **Resources > JMS > Queue connection factories**.
 - B. From the Scope drop-down list select the server as your scope, and click **New**.
 - C. Select the default messaging provider and click **OK**.
 - D. Under General Properties enter `webServicesReply_QCF` as the name (you **must** use `webServicesReply_QCF` for this field) and a JNDI name (for example `jms/webServicesReplyQCF`). If you want to use a custom name for the reply queue connection factory you have to change the reference alias in the `JMSServiceRouter` deployment descriptor. This reference is set up when you run the web service wizards. Thus if you decide to use a different JNDI name you have to go into this project and override the default setting.
 - E. In the Connection pane select the bus created earlier (`ws_test_Bus`) as the bus name and click **OK** to save the changes.
 15. A JMS activation specification is needed to bind the input queue and the listening message driven EJB:
 - A. Select **Resources > JMS > Activation specifications**.
 - B. From the Scope drop-down list select the server as your scope, and click **New**.
 - C. Select the default messaging provider and click **OK**.
-

D. Enter a name (for example `ws_test_JMSRouter`), and enter a JNDI name (for example `eis/ws_test_JMSRouter`).

In the Destination pane select **Queue** as the destination type, enter the destination JNDI name (`jms/ws_test_queue`), and select the bus name (`WS_test_Bus`).

E. Click **OK** to save the changes.

16. Once you have added the required connection factories and queues or topics, save your configuration. Stop and restart WebSphere Application Server and return to the development workspace.

HTTP and JMS transport methods

Web services created using the WebSphere® runtime environments support a JMS transport layer in addition to the existing HTTP transport. This allows web service clients and servers to communicate using JMS queues and topics instead of HTTP connections. Both one-way and synchronous two-way requests are supported.

Important: Applicable edition: Full profile

Restriction:

- A web service must be implemented as an EJB if it will be accessible through the JMS transport.
- Web services using JMS cannot be tested using the Web Services Explorer.

The benefits of using JMS as an alternative to HTTP include the following:

- Request and response messages are sent by way of reliable messaging.
- One-way requests allow client and server to be more loosely-coupled (the server does not have to be active when the client sends the one-way request).
- One-way requests can be sent to multiple servers simultaneously through the use of a topic.

If a web service is to be accessible on the JMS transport, then the corresponding WSDL document should include a JMS binding and a SOAP address which specifies a JMS endpoint URL string. A JMS binding is simply a **wSDL:binding** element which contains a **wSDLsoap:binding** element whose **transport** attribute ends in **soap/jms**, rather than the normal **soap/http** value. In addition to the JMS binding, a **wSDL:port** element which references the JMS binding should be included in the **wSDL:service** element within the WSDL document. This **wSDL:port** element should contain a **wSDLsoap:address** element whose **location** attribute specifies a JMS endpoint URL string.

You also need to decide on the names and types of JMS objects that your application will use. For example, you must decide whether your web service will receive its requests from a queue or a topic. You also must decide whether to use a secure destination (queue or topic). Finally, you will need to decide on the names for your destination, connection factory, and listener port. The following list provides an example of the names that might be used for the sample **StockQuote** Web service:

- Queue: **StockQuote_Q** (JNDI name: **jms/StockQuote_Q**)
- Inbound queue connection factory: **StockQuoteQCF** (JNDI name: **jms/StockQuoteQCF**)
- Outbound queue connection factory: **WebServicesReplyQCF** (JNDI name: **jms/WebServicesReplyQCF**). This is a reserved, case-sensitive name. You cannot give your outbound queue connection factory any other name.
- Listener port: **StockQuoteEJB_ListenerPort** (connection factory cannot be **jms/WebServicesReplyQCF**)

After creating your web service, you can run the Create Router Modules wizard to add a JMS endpoint (router module) for each web service-enabled EJB .jar contained in the EAR file. If you create the web service using the web services wizard, this is done for you automatically.

Parent topic: [Creating a JMS server](#)

Configuring the IBM® JRE to talk to a secured WebSphere® Application Server

Use these steps if you want to use the web services wizard to retrieve an HTTPS WSDL or if you want to use the Web Services Explorer against a secured WebSphere® Application Server. If you encounter an error similar to `Error opening socket: javax.net.ssl.SSLHandshakeException: unknown certificate` this task resolves the issue. This error occurs because WebSphere Application Server uses a security certificate for negotiating secured connections that other JRE-based applications do not normally share.

About this task

Important: Applicable edition: Full profile

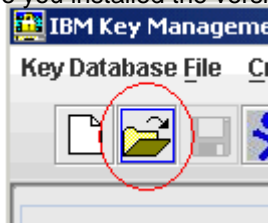
To configure your JRE to accept the WebSphere Application Server certificate:

Procedure

1. Start the iKeyman tool from your Eclipse JRE, which is in the following location within your WebSphere Application Server installation directory: `install_dir\java\jre\bin\ikeyman.exe`. The default installation locations for the servers:

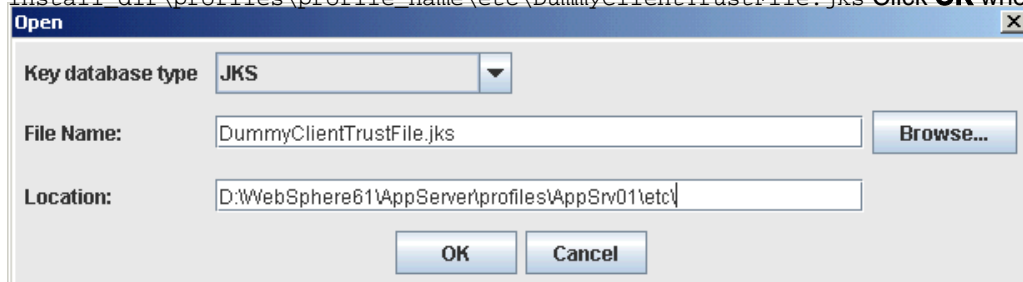
- WebSphere Application Server: `install_dir\java\jre\bin\ikeyman.exe`

Note: The `install_dir` directory is where you installed the version of WebSphere Application Server.



2. Click the **Open a key database file** icon:
3. In the window that opens, click **Browse** and locate the `DummyClientTrustFile.jks` in your WebSphere Application Server profile. The default location might be similar to

`install_dir\profiles\profile_name\etc\DummyClientTrustFile.jks` Click **OK** when you find the file.



4. You are prompted for a password. Enter `webAS`.
5. Select **Signer Certificates** from the list, and then select **default_signer** and click **Extract**.
6. Note the location and name of the certificate because it is needed in later steps. Click **OK** to save the file.
7. Click the **Open a key database file** icon again, and browse to the Eclipse JRE cacerts. This file is located here:
`install_dir\java\jre\lib\security\cacerts`.
8. When prompted for a password enter `changeit`.
9. Click **Add**, and browse to the file that you saved earlier. You must set the file types field to All Files. Click **OK** when the correct file is selected in the open window.
10. Enter a label for the certificate.

Results

The JRE can now accept the server certificate automatically. The certificate might restrict to the same host name on the certificate (the host name, including the domain).

Parent topic: [Configuring a workspace for web services development](#)

Problems working with a secured server using SSL connections

How to work around the `SSLSocketFactory` and `SSLHandshakeException` error messages when you are trying to communicate to a secured server by using a Secure Sockets Layer (SSL) connection within the workbench.

Important: Applicable edition: Full profile

While you are developing in the workbench, you might encounter the following `SSLSocketFactory` error message:

```
java.net.SocketException: java.lang.ClassNotFoundException: Cannot find the specified class
com.ibm.websphere.ssl.protocol.SSLSocketFactory
```

Here is a list of known tools where this `SSLSocketFactory` error message is displayed:

- Web services wizard retrieving an HTTPS WSDL
- Web Services Explorer running against a secured WebSphere® Application Server
- Deploying web services to the Axis runtime environment
- Installing Rational® Asset Manager with an SSL enabled WebSphere Application Server

The workaround for the `SSLSocketFactory` error message, is to look for the following line in the

`com.ibm.ws.ast.st.core.prefs` file available in the `x:\workspace`

`\.metadata\plugins\org.eclipse.core.runtime\settings`, where `x:\workspace` is the directory of your workspace.
`isUseIBMSSLSocketFactory=true`

Replace `true` with `false`. Then, restart the workbench. After you change the `isUseIBMSSLSocketFactory` property to `false`, you might encounter the following `SSLHandshakeException` error message:

```
javax.net.ssl.SSLHandshakeException: com.ibm.jsse2.util.g: No trusted certificate found
```

To resolve the `SSLHandshakeException` error message when the `isUseIBMSSLSocketFactory` property is set to `false`, see the [Configuring the IBM® JRE to talk to a secured WebSphere Application Server](#) topic for instructions for running the iKeyman tool to work around this problem.

Modifying the `isUseIBMSSLSocketFactory` property to `false` might cause the `SSLHandshakeException` error message to persist when you are connecting or switching for the first time to a secured WebSphere Application Server by using an

SSL connection within the workbench:
`Server.userException javax.net.ssl.SSLHandshakeException: com.ibm.jsse2.util.g: No trusted certificate found`

The workaround for this `SSLHandshakeException` error message, is to change back the `isUseIBMSSLSocketFactory` property to `true`. Look for the following line in the `com.ibm.ws.ast.st.core.prefs` file available in the `x:\workspace`

`\.metadata\plugins\org.eclipse.core.runtime\settings`, where `x:\workspace` is the directory of your workspace.
`isUseIBMSSLSocketFactory=false`

Replace `false` with `true`. Then, restart the workbench.

Parent topic: [Configuring a workspace for web services development](#)

Parent topic: [Specifying administrative settings to a secured WebSphere Application Server](#)

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Related tasks:

[Specifying administrative settings to a secured WebSphere Application Server](#)

[Manually exchanging signer certificates to establish a trust between the workbench and the server](#)

Importing and creating resources for web services

In order to create a web service or web service client you must find, import, or create the resources that the web service or client will use.

If you are creating a bottom-up web service you must import or create a Java™ bean or EJB.

- For information on developing EJBs, refer to the [Developing enterprise applications](#) section of the online help.

If you are creating a top-down web service you must create or import a WSDL file.

- To import a WSDL file refer to [Importing a WSDL file](#).

- To create a WSDL file, refer to [Creating a WSDL file](#).

If you are creating a web service client you must have an existing web service in your workspace, or discover an existing web service and import its WSDL file.

- To discover a web service refer to [Discovering web services](#).

Developing Web services and clients

You can create web services and clients by using the web services wizards, annotations, Ant tasks, or command-line tools.

- **Methods of creating Web services and clients**

Web services and clients can be created by various methods such as annotations, Ant tasks, or web services wizards.

- **Creating web services and clients by using the web service wizards**

Use web services tools to discover, create, and publish web services that are created from Java™ beans, enterprise beans, and WSDL files. You can create web services by using a top-down approach (which starts with a WSDL file) or a bottom-up approach (which starts with a Java bean or EJB).

- **Creating web services by using annotations**

With the Java API for XML-Based web services, you can use annotations in your Java code to simplify creating web services.

- **Creating web services and clients with Ant tasks or command line tools**

If you prefer not to use the web service wizards, you can use Ant tasks or command line tools to create web services using the IBM® WebSphere® runtime environments or Axis runtime environment.

Parent topic: [Developing web service applications](#)

Methods of creating Web services and clients

Web services and clients can be created by various methods such as annotations, Ant tasks, or web services wizards. The following tables outline the various methods of creating web services and clients, sorted by the type of artifact you are using to create the service or client.

Table 1. Web services created from Java™ beans

	JAX-WS
WebSphere® Application Server V7.0 or later	Web services wizard Annotations Ant task
WebSphere Application Server V8.5.5 Liberty Profile	Web services wizard Annotations Ant task

Table 2. Web services created from EJB v2.x enterprise beans

	JAX-WS
WebSphere Application Server V7.0 or later	Not supported
WebSphere Application Server V8.5.5 Liberty Profile	Not supported

Table 3. Web services created from EJB 3.x enterprise beans

	JAX-WS
WebSphere Application Server V7.0 or later	Annotations
WebSphere Application Server V8.5.5 Liberty Profile	Annotations

Table 4. Web services created from WSDL files

	JAX-WS
WebSphere Application Server V7.0 or later	Web services wizard Ant task
WebSphere Application Server V8.5.5 Liberty Profile	Web services wizard Ant task

Table 5. Web service clients created from WSDL files

	JAX-WS
WebSphere Application Server V7.0 or later	Web services wizard Ant task
WebSphere Application Server V8.5.5 Liberty Profile	Web services wizard Ant task

Parent topic: [Developing Web services and clients](#)

Creating web services and clients by using the web service wizards

Use web services tools to discover, create, and publish web services that are created from Java™ beans, enterprise beans, and WSDL files. You can create web services by using a top-down approach (which starts with a WSDL file) or a bottom-up approach (which starts with a Java bean or EJB).

- IBM WebSphere JAX-WS runtime environment

Java API for XML-Based Web Services (JAX-WS) is the next generation web services programming model. Using JAX-WS, development of web services and clients is simplified with greater platform independence for Java applications by the use of dynamic proxies and Java annotations.

- Creating a Java bean skeleton from a WSDL document using the WebSphere JAX-WS runtime environment

The web service wizard assists you in creating a skeleton bean from an existing WSDL document. The skeleton bean contains a set of methods that correspond to the operations described in the WSDL document. When the bean is created, each method has a trivial implementation that you replace by editing the bean.

- Creating an enterprise bean (EJB) skeleton from a WSDL document using the WebSphere JAX-WS runtime environment

The web service wizard assists you in creating a skeleton bean from an existing WSDL document. The skeleton bean contains a set of methods that correspond to the operations described in the WSDL document. When the bean is created, each method has a trivial implementation that you replace by editing the bean.

- Creating a web service from a Java bean using the IBM WebSphere JAX-WS runtime environment

The web service wizard assists you in creating a new web service, configuring it for deployment, and deploying the web service to a server. Once your web service is deployed, the wizard assists you in generating the client proxy and sample application to test the web service. When you have completed testing, you can publish your web service to a UDDI Business Registry using the Export wizard.

- Creating a JAX-RS web service

- Creating web service clients

You can create a web service client for a web service in your workspace or a remote service.

Parent topic: [Developing Web services and clients](#)

Creating a Java bean skeleton from a WSDL document using the WebSphere JAX-WS runtime environment

The web service wizard assists you in creating a skeleton bean from an existing WSDL document. The skeleton bean contains a set of methods that correspond to the operations described in the WSDL document. When the bean is created, each method has a trivial implementation that you replace by editing the bean.

Before you begin

Prerequisites:

- If you are using WebSphere® Application Server, it is strongly suggested that you start the server before running the web service wizard because it may take several minutes to start the WebSphere Application Server depending on the speed of your computer. To start the server, select it in the Servers view (**Window > Show View > Servers**), right-click and select **Start**.
- Create or discover and import a WSDL document into a project. You can use a WSDL file that contains a service element.

About this task

To create a skeleton Java™ bean from a WSDL document by using the WebSphere JAX-WS runtime environment:

Procedure

1. Switch to the Java EE perspective (**Window > Open Perspective > Java EE**).
2. In the Enterprise Explorer view, select the WSDL file that you have created or imported.
3. Click **File > New > Other**. Select **Web Services** to display the various web service wizards. Select the **Web Service** wizard. Click **Next**.
4. Web Services page: select **Top down Java bean Web service** as your web service type. You can also choose to do the following:
 - A. Select the stages of web services development that you want to complete using the slider. This will set several default values on the remaining wizard panels:
 - Develop: this will develop the WSDL definition and implementation of the web service. This includes such tasks as creating the modules which will contain the generated code, WSDL files, deployment descriptors, and Java files when appropriate.
 - Assemble: this ensures the project that will host the web service or client gets associated to an EAR when required by the target application server.
 - Deploy: this will create the deployment code for the service.
 - Install: this will install and configure the Web module and EARs on the target server. If any changes to the endpoints of the WSDL file are required they will be made in this stage.
 - Start: this will start the server once the service has been installed on it.
 - Test: this will provide various options for testing the service, such as using the Generic Services Client, Web Service Explorer or sample JSPs.
 - B. Select your Server runtime: the default server is displayed. If you want to deploy your service to a different server click the server link and specify a different server. This task supports the following server runtime environments:
 - WebSphere Application Server V7.0 or later
 - WebSphere Application Server Liberty Profile V8.5.5

- C. Select your Web service runtime: the default runtime is displayed. To deploy your service to the IBM® WebSphere JAX-WS runtime click the runtime link and select it in the window that opens.
- D. Select the Service project and Service EAR project: the project selected in your workspace is displayed. Only web projects with the Java 5.0, Java 6.0, or Java 7.0 facet enabled are supported. To select a different project and EAR click on the project link, or enter a name and allow the wizard to create a project for you. Ensure that the project selected as the Client Web Project is different from the Service Web Project, or the service will be overwritten by the client's generated artifacts. For JAX-WS web services, the server and client projects can share the same EAR.
- E. If you want to create a client, select the type of proxy to be generated and repeat steps 1 through 4 for the client. The client can be created later using the steps outlined in: [Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment](#)
- F. Monitor the Web service: this will send the web service traffic through the TCP/IP Monitor, which allows you to watch the SOAP traffic generated by the web service and to test this traffic for WS-I compliance. Alternately you can manually set up a TCP/IP monitor as described in [Using the TCP/IP Monitor to test web services](#).
5. WebSphere JAX-WS Top Down Web Service Configuration page:
- Output folder: Enter the location where the generated Java skeleton is generated or accept the default.
 - Target package: Enter the package name for the generated Java or accept the default.
 - Enable wrapper style: Enables wrapper style mapping from WSDL to Java. For WSDL documents that implement a document/literal wrapped pattern, a root element is declared in the XML schema and is used as an operation wrapper for a message flow. Separate wrapper element definitions exist for both the request and the response. The element whose name is the same as the operation (the wrapper element) is broken apart so that each of its content elements becomes a parameter of the generated Java method signature.
 - Generate serializable JAXB classes: In WebSphere Application Server and WebSphere(r) Application Server Liberty Profile V8.5.5 when you enable the Java 6 facet, you can choose to generate JAXB classes, which implement `java.io.Serializable`. Classes that do not implement this interface do not have their states serialized or deserialized.
 - Enable MTOM support: If you select this check box the SOAP Message Transmission Optimization Mechanism is enabled to optimize the transmission of binary content. For more information on MTOM read [MTOM Overview](#)
 - Version of JAX-WS code to be generated: WebSphere Application Server V7.0, supports JAX-WS 2.0 or 2.1 compliant code. WebSphere Application Server V8.0, V8.5, and WebSphere(r) Application Server Liberty Profile V8.5.5, supports JAX-WS 2.0, 2.1, or 2.2 compliant code.
 - Copy WSDL to project: Select to copy the WSDL file into the service project, or the required WSDL is generated dynamically by the runtime when needed. This is a convenient option if you plan to create the client later or publish the WSDL for other users.
 - Generate serializable JAXB classes: In WebSphere Application Server and WebSphere(r) Application Server Liberty Profile V8.5.5 when you enable the Java 6 facet, you can choose to generate JAXB classes, which implement `java.io.Serializable`. Classes that do not implement this interface do not have their states serialized or deserialized.
 - Specify JAX-WS or JAXB binding files: If you created JAX-WS or JAXB custom binding files, select this check box to use them to create this web service. If this check box is selected, the next page of the wizard allows you to browse to the custom binding declaration files.
 - Customize service implementation class name: Select this check box if you want to change the default port name to service implementation class name mapping.
 - Generate schema library: allows you to generate schema code into a schema library. If you select a schema library, on a later page you can customize the mappings between XSD documents or groups of XSD documents to a project.
 - Generate web service deployment descriptor: For JAX-WS web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer needed. Selecting this check box generates them.
 - Allow vendor extensions: If you select this check box, the `-extension` option is added to the **wsgen** or **wsimport** command. The `-extension` option specifies whether to enable or accept custom extensions for functionality not specified by the JAX-WS specification. Use of the extensions can result in applications that are not portable or do not
-

interoperate with other implementations. For details, see [wsген command for JAX-WS applications](#) and [wsimport command for JAX-WS applications](#) topics available in the Information Center for WebSphere Application Server.

6. Depending on the options that are chosen on the previous page, you might be prompted to customize your web service implementation on the following pages:
 - A. Custom Binding Declarations page: If you selected to specify JAX-WS or JAXB binding files, you can browse to the files on this page.
 - B. Service Implementation Configuration page: For each port defined in the WSDL file, you can enter a name for the service implementation class or accept the default.
 - C. Schema project configuration page: If you selected to generate a schema library, use this page to customize the mappings between XSD documents and projects. This feature works if all the schema dependencies are in different targetNamespaces; otherwise there are compilation errors in ObjectFactory.java.
7. Test Web Service page: If you selected to test the web service, select the test facility for the generated web service and click **Launch**. This opens the web service in the Web Services Explorer or Generic Services Client. Select the operation that you want to test, enter the required information, and click **Go**. The result is shown in the Status pane. Click **Next**.
8. WebSphere JAX-WS web services Client Configuration page: if you selected to generate a client, you can specify the following options for it on this page:
 - Output folder: The location where the client code is generated
 - Target package: The web services client wizard generates a number of Java files. By default it creates a package name that is based on the namespace that is specified in the WSDL file. To override this default behavior, you can specify your own package name.
 - Generate portable client: Select this check box to move your web service client code from one machine to another or from one instance of WebSphere Application Server or WebSphere Application Server Liberty Profile V8.5.5 to another. If this option is selected, the WSDL document and all the XML schema and other WSDL documents that it depends upon is copied into the client project under WEB-INF/wsdl and a `file:relativeURL` pointing to this copy is then injected into the JAX-WS Service class's static initialization block.
 - Enable asynchronous invocation for generated client: If you select to enable an asynchronous client, for each method in the web service two more methods are created. These are polling and callback methods, which allow the client to function asynchronously.
 - Specify JAX-WS or JAXB binding files: If you created JAX-WS or JAXB custom binding files, select this check box to use them to create the web service client. If this check box is selected, the next page of the wizard will allow you to browse to the custom binding declaration files.
 - Customize client proxy class name: Select this check box if you want to change the default port name to service implementation class name mapping
9. Custom Binding Declarations page: If you selected to specify JAX-WS or JAXB binding files, you can browse to the files on this page.
10. Service implementation configuration page: Proxy Generation Configuration page: For each port defined, you can enter a name for the proxy class or accept the default.
11. Web Service Client Test page:
 - Test the generated proxy: If selected, the sample client is started in a web browser that allows you to test the function.
 - Select your test facility. You can test the generated proxy in the Generic Service Client or the Web Service Explorer, or you can generate and use the sample JAX-WS 2.0 JSPs.
 - Folder: If you selected to test the proxy through a JSP, you can select the folder where the JSP is located, and you can select the methods that are included in the JSP.
 - Methods: Select the methods to expose. If you selected to enable asynchronous invocation, the asynchronous methods are listed as well.
 - Run test on server: Starts the server for you automatically.

Results

After the web service is created, the following might occur depending on the options you selected:

- The implementation bean is opened in the Java editor. This contains skeleton code that can be used to test the basic function of the web service, or you can implement the required business logic for your service before you test it.
- If you selected to test the generated proxy by using JAX-WS 2.0 JSPs, the proxy is started in a web browser at the following URL:`http://localhost:port/WebProjectClient/sampleBeanName/WebServiceName/TestClient.jsp` You can use this sample application to test the web service by selecting a method, entering a value for the method, and clicking **Invoke**. The result of the method is displayed in the results pane. Since this is a Java bean skeleton with trivial method implementation, a value of **-3** is returned.
- If you selected to test the generated proxy by using the Universal Test Client, it is started in a browser window at a URL similar to the following: `http://localhost:port/UTC/preload?object=BeanPackage.BeanServiceProxy`. Under **Objects**, expand the proxy stub to display the methods of the web service. Click the method that you want to test, enter a value in the **Parameters** pane, and click **Invoke**. The result is generated and displayed.
- If you selected to test the web service by using the Web Services Explorer, the Explorer opens. Select the operation that you want to test, enter the required information, and click **Go**. The result is displayed in the **Status** pane.

Parent topic:[Creating web services and clients by using the web service wizards](#)

Related concepts:

[Java API for XML based web services](#)

[JAXB](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Web services runtime environments](#)

Creating an enterprise bean (EJB) skeleton from a WSDL document using the WebSphere JAX-WS runtime environment

The web service wizard assists you in creating a skeleton bean from an existing WSDL document. The skeleton bean contains a set of methods that correspond to the operations described in the WSDL document. When the bean is created, each method has a trivial implementation that you replace by editing the bean.

Before you begin

Prerequisites:

- If you are using WebSphere® Application Server, it is strongly suggested that you start the server before running the web service wizard because it may take several minutes to start the WebSphere Application Server depending on the speed of your computer. To start the server, select it in the Servers view (**Window > Show View > Servers**), right-click and select **Start**.
- Create or discover and import a WSDL document into a project. You can only use a WSDL file that contains a service element. **Note:** SOAP over JMS web services are supported for WebSphere Application Server V8.0 or later; however, support is not available for WebSphere Application Server Liberty Profile V8.5.5. If you want to use SOAP over JMS, your WSDL file must use one of the following transport attribute URI values:
 - <http://www.w3.org/TR/soapjms/>
 - <http://www.w3.org/2010/soapjms/>For more information about SOAP over JMS declarations in WSDL files, refer to the specifications at: [SOAP over Java™ Message Service 1.0: Working Draft](#)
- Create an EJB project and add it to a new EAR project.
- Create an empty Web project if you are using SOAP over HTTP as your transport methods, or an EJB project if you are using SOAP over JMS as your transport method, to act as the router project. The project you create must be added to the same EAR as the EJB project containing the enterprise bean. For more information on router modules refer to [Creating router modules](#).
- If you want to use SOAP over JMS, you must create a server and server configuration for JMS as described in [Create a server and server configuration for SOAP over JMS](#).

About this task

To create a skeleton Java bean from a WSDL document using the WebSphere JAX-WS runtime environment:

Procedure

1. Switch to the Java EE perspective (**Window > Open Perspective > Java EE**).
2. In the Enterprise Explorer view, select the WSDL file you have created or imported.
3. Click **File > New > Other**. Select **Web Services** in order to display the various Web service wizards. Select the **Web Service** wizard. Click **Next**.
4. Web Services page: select **Top down EJB Web service** as your Web service type. You can optionally choose to do the following:
 - A. Select the stages of Web services development that you want to complete using the slider. This will set several default values on the remaining wizard panels:
 - Develop: this will develop the WSDL definition and implementation of the Web service. This includes such tasks as creating the modules which will contain the generated code, WSDL files, deployment descriptors, and Java files when appropriate.

- Assemble: this ensures the project that will host the Web service or client gets associated to an EAR when required by the target application server.
 - Deploy: this will create the deployment code for the service.
 - Install: this will install and configure the Web module and EARs on the target server. If any changes to the endpoints of the WSDL file are required they will be made in this stage.
 - Start: this will start the server once the service has been installed on it.
 - Test: this will provide various options for testing the service, such as using the Generic Services Client, Web Service Explorer or sample JSPs.
- B. Select your server: the default server is displayed. If you want to deploy your service to a different server click the server link and specify a different server.
- C. Select your runtime: the default runtime is displayed. To deploy your service to the IBM® JAX-WS runtime click the runtime link and select it in the window that opens. This task supports the following server runtime environments:
- WebSphere Application Server V7.0 or later
 - WebSphere Application Server Liberty Profile V8.5.5
- D. Select the service project: the project selected in your workspace is displayed. Only EJB projects are supported. To select a different project and EAR click on the project link, or enter a name and allow the wizard to create a project for you. Ensure that the project selected as the Client Web Project is different from the Service Web Project, or the service will be overwritten by the client's generated artifacts. For JAX-WS Web services, the server and client projects can share the same EAR.
- E. If you want to create a client, select the type of proxy to be generated and repeat the steps for the client. The client can be created later using the steps outlined in: [Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment](#)
- F. Monitor the Web service: this will send the Web service traffic through the TCP/IP Monitor, which allows you to watch the SOAP traffic generated by the Web service and to test this traffic for WS-I compliance. Alternately you can manually set up a TCP/IP monitor as described in [Using the TCP/IP Monitor to test web services](#).
- G. Publish the Web service.
5. WebSphere JAX-WS Top Down EJB Web Service Configuration page:
- Output folder: Enter the location where the generated EJB skeleton will be generated or accept the default.
 - Target package: Enter the package name for the generated Java or accept the default.
 - EJB Web service binding: select if your Web service will use HTTP or JMS bindings.
 - Enable wrapper style: Enables wrapper style mapping from WSDL to Java. For WSDL documents that implement a document/literal wrapped pattern, a root element is declared in the XML schema and is used as an operation wrapper for a message flow. Separate wrapper element definitions exist for both the request and the response. More simply, the element whose name is the same as the operation (the wrapper element) is broken apart so that each of its content elements becomes a parameter of the generated Java method signature.
 - Enable MTOM support: If you select this check box the SOAP Message Transmission Optimization Mechanism will be enabled to optimize the transmission of binary content. For more information on MTOM refer to: [MTOM Overview](#)
 - Version of JAX-WS code to be generated: WebSphere Application Server V7.0, supports JAX-WS 2.0 or 2.1 compliant code. WebSphere Application Server V8.0, V8.5, and WebSphere(r) Application Server Liberty Profile V8.5.5, supports JAX-WS 2.0, 2.1, or 2.2 compliant code.
 - Copy WSDL to project: Select this to copy the WSDL file into the service project, or the required WSDL will be generated dynamically by the runtime when needed. This is a convenient option if you plan to create the client at a later time or publish the WSDL for other users.
 - Generate serializable JAXB classes: In WebSphere Application Server and WebSphere(r) Application Server Liberty Profile V8.5.5 when you enable the Java 6 facet, you can choose to generate JAXB classes which implement `java.io.Serializable`. Classes that do not implement this interface will not have any of their state serialized or deserialized.

- Specify JAX-WS or JAXB binding files: If you have created JAX-WS or JAXB custom binding files, select this check box to use them to create this Web service. If this is selected the next page of the wizard will allow you to browse to the custom binding declaration files.
 - Customize service implementation class name: Select this check box if you want to change the default port name to service implementation class name mapping.
 - Generate schema library: this allows you to generate schema code into a schema library. If you select a schema library, on a later page you can customize the mappings between XSD documents or groups of XSD documents to a project.
 - Generate Web service deployment descriptor: For JAX-WS Web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer required. Selecting this checkbox will generate them.
 - Allow vendor extensions: If you select this check box, the `-extension` option is added to the **wsgen** or **wsimport** command. The `-extension` option specifies whether to enable or accept custom extensions for functionality not specified by the JAX-WS specification. Use of the extensions can result in applications that are not portable or do not interoperate with other implementations. For details, see [wsgen command for JAX-WS applications](#) and [wsimport command for JAX-WS applications](#) topics available in the Information Center for WebSphere Application Server.
6. Depending on the options chosen on the previous page, you may be prompted to customize your Web service implementation on the following pages:
- A. WebSphere JAX-WS JMS Binding Configuration page: if you selected for your Web service to use JMS as its transport method, you will need to enter information about the binding on this page.
- JMS destination: Select either queue or topic as your destination type.
 - Destination JNDI name: The JNDI name of the destination queue or topic.
 - JMS connection factory: The JNDI name of the connection factory.
 - Initial context factory name: The name of the initial context factory to use. This is mapped to the `java.naming.factory.initial` property.
 - JNDI provider URL: This is mapped to the `java.naming.provider.url` property.
 - Delivery mode: Indicates whether the request message should be persistent or not.
 - Request message lifetime: The lifetime in milliseconds of the request message. A value of 0 indicates an infinite lifetime.
 - JMS request message priority: The JMS priority associated with the request message. 0 is the lowest priority and 9 is the highest priority.
 - Reply to name: The name of the port component to which the request will be dispatched.
 - Connection factory user ID: The user ID to be used to gain access to the connection factory.
 - Connection factory password: The password to be used to gain access to the connection factory.
 - Use SOAP/JMS interoperability protocol: Selecting this ensures that the Web service complies with the protocol outlined by the W3C SOAP-JMS Binding Working Group. For more information refer to this site: [Charter of the SOAP-JMS Binding Working Group](#)
 - Define custom JMS properties: Selecting this will launch a page enabling you to enter property value pairs.
- B. Custom Binding Declarations page: If you have selected to specify JAX-WS or JAXB binding files you can browse to the files on this page.
- C. Service Implementation Configuration page: For each port defined in the WSDL file, you can enter a name for the service implementation class or accept the default.
- D. Schema project configuration page: If you selected to generate a schema library use this page to customize the mappings between XSD documents and projects. This feature will only work if all the schema dependencies are in different targetNamespaces; otherwise there will be compilation errors in `ObjectFactory.java`.
7. WebSphere JAX-WS Router Project Configuration page: This page displays when targeting WebSphere Application Server runtime environment, but not WebSphere Application Server Liberty Profile V8.5.5.
- HTTP router: enter the name of the project you want used as the router module. A router module provides a Web

- service endpoint for a particular transport.
 - JMS router: enter the name of the project you want used as the router module. A router module provides a Web service endpoint for a particular transport.
 - MDB deployment mechanism: select from the JMS activation spec or the listener port as your message-driven bean deployment mechanism.
 - ActivationSpec JNDI name: type the JNDI name of the activation specification that is to be used to deploy this message-driven bean. This name must match the name of an activation specification that you define to WebSphere Application Server
 - Listener input port name: enter the listener port name of the router module.
8. Test Web Service page: If you selected to test the web service from step 4a, select the test facility for the generated web service and click **Launch**. This will open the web service in the Web Services Explorer or Generic Services Client. Note that you can only use the Web Services Explorer to test web services which use HTTP transports. Select the operation you want to test, enter the required information, and click **Go**. The result will display in the Status pane. Click **Next**.
9. WebSphere JAX-WS Web Services Client Configuration page: if you have selected to generate a client, you can specify the following options for it on this page:
- Output folder: This is the location where the client code will be generated
 - Target package: The web services client wizard generates a number of Java files. By default it will create a package name based on the namespace specified in the WSDL file. To override this default behavior you can specify your own package name.
 - Generate portable client: Selecting this checkbox would allow you to move your web service client code from one machine to another or from one instance of WebSphere Application Server or WebSphere(r) Application Server Liberty Profile V8.5.5 to another. If this option is selected, the WSDL document and all the XML Schema and other WSDL documents that it depends upon will be copied into the client project under WEB-INF/wsdl and a `file:relativeURL` pointing to this copy will then be injected into the JAX-WS Service class's static initialization block.
 - Enable asynchronous invocation for generated client: If you select to enable an asynchronous client, for each method in the web service two additional methods are created. These are polling and callback methods which allow the client to function asynchronously.
 - Specify JAX-WS or JAXB binding files: If you have created JAX-WS or JAXB custom binding files, select this check box to use them to create the web service client. If this is selected the next page of the wizard will allow you to browse to the custom binding declaration files.
 - Customize client proxy class name: Select this check box if you want to change the default port name to service implementation class name mapping
 - Generate web service deployment descriptor: For JAX-WS web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer required. Selecting this checkbox will generate them.
 - Version of JAX-WS code to be generated: For WebSphere Application Server V7.0, you have the option of generating JAX-WS 2.0 or 2.1 compliant code. For WebSphere Application Server V8.0, V8.5, and WebSphere Application Server Liberty Profile V8.5.5, you have the option of generating JAX-WS 2.0, 2.1, or 2.2 compliant code.
10. Custom Binding Declarations page: If you have selected to specify JAX-WS or JAXB binding files you can browse to the files on this page.
11. Service implementation configuration page: Proxy Generation Configuration page: For each port defined, you can enter a name for the proxy class or accept the default.
12. Web Service Client Test page:
- Test the generated proxy: If selected, the sample client will be launched in a Web browser enabling you to test the functionality.
 - Select your test facility. You can test the generated proxy in the Generic Service Client or the Web Service Explorer, or you can generate and use the sample JAX-WS 2.0 JSPs.
-

- Folder: If you selected to test the proxy through a JSP, you can select the folder where the JSP will be located, and you can select the methods that will be included in the JSP.
- Methods: Select the methods to expose. If you selected to enable asynchronous invocation, the asynchronous methods should be listed as well.
- Run test on server: this will start the server for you automatically.

Results

After the web service has been created, the following may occur depending on the options you selected:

- The implementation bean will be opened in the Java editor. This contains skeleton code that can be used to test the basic function of the web service, or you can implement the required business logic for your service before testing it.
- If you have selected to test the generated proxy using JAX-WS 2.0 JSPs, the proxy is launched in a Web browser at the following URL:`http://localhost:port/WebProjectClient/sampleBeanName/WebServiceName/TestClient.jsp` You can use this sample application to test the web service by selecting a method, entering a value for the method, and clicking **Invoke**. The result of the method will display in the results pane. Since this is a Java bean skeleton with trivial method implementation, a value of -3 will be returned.
- If you have selected to test the generated proxy using the Universal Test Client, it will be launched in a browser window at a URL similar to the following: `http://localhost:port/UTC/preload?object=BeanPackage.BeanServiceProxy`. Under Objects, expand the proxy stub to display the methods of the web service. Click the method you want to test, enter a value in the Parameters pane, and click **Invoke**. The result will be generated.
- If you have selected to test the web service using the Web Services Explorer, the Explorer will open. Select the operation you want to test, enter the required information, and click **Go**. The result will display in the Status pane.

Parent topic: [Creating web services and clients by using the web service wizards](#)

Creating a web service from a Java bean using the IBM WebSphere JAX-WS runtime environment

The web service wizard assists you in creating a new web service, configuring it for deployment, and deploying the web service to a server. Once your web service is deployed, the wizard assists you in generating the client proxy and sample application to test the web service. When you have completed testing, you can publish your web service to a UDDI Business Registry using the Export wizard.

Before you begin

- If you are using WebSphere® Application Server, it is strongly suggested that you start the server before running the web service wizard because it may take several minutes to start the WebSphere Application Server depending on the speed of your computer. To start the server, select it in the Servers view (**Window > Show View > Servers**), right-click and select **Start**.
- Create or import a bean into the Java™ source folder of a web project or Java project. To import, select the **Java Resources: src** folder, and from the File menu select **Import > General > File System** and browse to where your package is.

About this task

To create a web service from a bean using the IBM® WebSphere JAX-WS runtime environment:

Procedure

1. Switch to the Java EE perspective (**Window > Open Perspective > Java EE**).
2. In the Enterprise Explorer view, select the bean that you created or imported into the source folder of your Web project.
3. Click **File > New > Other**. Select **Web Services** in order to display the various web service wizards. Select the **Web Service** wizard. Click **Next**.
4. In the **Web Services page** page, select **Bottom up Java bean Web service** as your web service type. You can optionally choose to do the following:
 - A. Select the stages of web services development that you want to complete using the slider. This will set several default values on the remaining wizard panels:
 - Develop: this will develop the WSDL definition and implementation of the web service. This includes such tasks as creating the modules which will contain the generated code, WSDL files, deployment descriptors, and Java files when appropriate.
 - Assemble: this ensures the project that will host the web service or client gets associated to an EAR when required by the target application server.
 - Deploy: this will create the deployment code for the service.
 - Install: this will install and configure the Web module and EARs on the target server. If any changes to the endpoints of the WSDL file are required they will be made in this stage.
 - Start: this will start the server once the service has been installed on it.
 - Test: this will provide various options for testing the service, such as using the Generic Services Client, Web Service Explorer or sample JSPs.
 - B. Select your Server runtime: the default server is displayed. If you want to deploy your service to a different server click the server link and specify a different server. This task supports the following server runtime environments:
 - WebSphere Application Server V7.0 or later
 - WebSphere Application Server Liberty Profile V8.5.5

- C. Select your Web service runtime: the default runtime is displayed. To deploy your service to the IBM WebSphere JAX-WS runtime click the runtime link and select it in the window that opens.
- D. Select the Service project and Service EAR project: the project selected in your workspace is displayed. Only web projects with the Java 5.0, Java 6.0, or Java 7.0 facet enabled are supported. To select a different project and EAR click on the project link, or enter a name and allow the wizard to create a project for you. Ensure that the project selected as the Client Web Project is different from the Service Web Project, or the service will be overwritten by the client's generated artifacts. For JAX-WS web services, the server and client projects can share the same EAR.
- E. If you want to create a client, select the type of proxy to be generated and repeat steps 1 through 4 for the client. The client can be created later using the steps outlined in: [Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment](#)
- F. Monitor the Web service: this will send the web service traffic through the TCP/IP Monitor, which allows you to watch the SOAP traffic generated by the web service and to test this traffic for WS-I compliance. Alternately you can manually set up a TCP/IP monitor as described in [Using the TCP/IP Monitor to test web services](#).
5. WebSphere JAX-WS Bottom Up Web Service Configuration:
- Delegate class name: Enter the class name of the delegate Java implementation or accept the default. The delegate class is a wrapper that contains all the methods from the Java bean as well as the JAX-WS annotation the runtime recognizes as a web service. It is generated into the same package as the original bean.
 - Java to WSDL mapping style: The style defines encoding style for messages sent to and from the web service. The parameter style determines whether the method's parameters represent the entire message body or whether parameters are elements wrapped inside a top-level element named after the operation. The valid combinations are RPC, Document/Wrapped, or Document/Bare.
 - Generate `@WebParam` annotations for method parameters: If you select this check box, the parameters in a method declaration is going to have the `@WebParam` annotation set for the respective arguments. The `@WebParam` annotation customizes the mapping of an individual parameter to a web service message part and XML element. For more details about `@WebParam` annotation, search for `javax.jws.WebParam` in the [JAX-WS annotations](#) topic available in the Information Center for WebSphere Application Server. **Tip:** You can enable this option by default in the Preference page by going to the toolbar and selecting **Window > Preferences > WebSphere > JAX-WS Code Generation**. Select **Generate @WebParam annotations for method parameters** check box under the **Bottom Up** section.
 - Enable SOAP 1.2 binding: If you do not select this, SOAP 1.1 bindings will be used. For additional information on the differences between SOAP 1.1 and 1.2, refer to: [SOAP Overview](#)
 - Enable MTOM support: If you select this check box the SOAP Message Transmission Optimization Mechanism will be enabled to optimize the transmission of binary content. For more information on MTOM refer to: [MTOM Overview](#)
 - Generate WSDL file into the project: Because the annotations in the delegate class are used to tell the runtime that the bean is a web service, a static WSDL file is no longer generated into your project automatically. The runtime can dynamically generate a WSDL file from the information in the bean. Select this to generate a static WSDL file for the web service. This is a convenient option if you plan to create the client at a later time or publish the WSDL for other users. You can configure the WSDL using the next page of the wizard.
 - Generate Web Service Deployment Descriptor: For JAX-WS web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer required. Selecting this checkbox will generate them.
 - Allow vendor extensions: If you select this check box, the `-extension` option is added to the **wsgen** or **wsimport** command. The `-extension` option specifies whether to enable or accept custom extensions for functionality not specified by the JAX-WS specification. Use of the extensions can result in applications that are not portable or do not interoperate with other implementations. For details, see [wsgen command for JAX-WS applications](#) and [wsimport command for JAX-WS applications](#) topics available in the Information Center for WebSphere Application Server.
- Note:** If the bean already has a `@javax.jws.WebService` annotation, most of the fields on this page will be disabled because the wizard does not need to generate a delegate bean for you. You will only be able to select to generate a

WSDL file. If you have added only the `@javax.jws.WebService` to your Java bean and want to enable other options such as SOAP 1.2 binding or MTOM, you should exit the wizard and either remove the annotation or proceed to create the web service using the annotations documentation. The wizard will not allow you to append new annotations to a pre-existing partially annotated bean.

6. WebSphere JAX-WS WSDL Interface Configuration page: If on the previous page you selected to generate a WSDL file, this page will display, allowing you to configure the generated WSDL file. You can configure the following:
 - WSDL target namespace
 - WSDL service name
 - WSDL port name
7. Test Web Service page: If you selected to test the web service, select the test facility for the generated web service and click **Launch**. This will open the web service in the Web Services Explorer or Generic Services Client. Select the operation you want to test, enter the required information, and click **Go**. The result will display in the Status pane. Click **Next**.
8. WebSphere JAX-WS Web Services Client Configuration page: if you have selected to generate a client, you can specify the following options for it on this page:
 - Output folder: This is the location where the client code will be generated
 - Target package: The web services client wizard generates a number of Java files. By default it will create a package name based on the namespace specified in the WSDL file. To override this default behavior you can specify your own package name.
 - Generate portable client: Selecting this checkbox would allow you to move your web service client code from one machine to another or from one instance of WebSphere Application Server to another. If this option is selected, the WSDL document and all the XML Schema and other WSDL documents that it depends upon will be copied into the client project under WEB-INF/wsdl and a `file:relativeURL` pointing to this copy will then be injected into the JAX-WS Service class's static initialization block.
 - Enable asynchronous invocation for generated client: If you select to enable an asynchronous client, for each method in the web service two additional methods will be created. These are polling and callback methods which allow the client to function asynchronously.
 - Specify JAX-WS or JAXB binding files: If you have created JAX-WS or JAXB custom binding files, select this checkbox to use them to create the web service client. If this is selected the next page of the wizard will allow you to browse to the custom binding declaration files.
 - Customize client proxy class name: Select this checkbox if you want to change the default port name to service implementation class name mapping
9. Custom Binding Declarations page: If you have selected to specify JAX-WS or JAXB binding files you can browse to the files on this page.
10. Proxy Generation Configuration page: For each port defined, you can enter a name for the proxy class or accept the default.
11. Web Service Client Test page:
 - Test the generated proxy: If selected, the sample client will be launched in a Web browser enabling you to test the functionality.
 - Select your test facility. You can test the generated proxy in the Generic Service Client or the Web Service Explorer, or you can generate and use the sample JAX-WS 2.0 JSPs.
 - Folder: If you selected to test the proxy through a JSP, you can select the folder where the JSP will be located, and you can select the methods that will be included in the JSP.
 - Methods: Select the methods to expose. If you selected to enable asynchronous invocation, the asynchronous methods should be listed as well.
 - Run test on server: this will start the server for you automatically.

Results

After the web service has been created, the following may occur depending on the options you selected:

- If you have selected to test the generated proxy using web service JSPs, the proxy is launched in a Web browser at the following URL: `http://localhost:port/WebProjectClient/sampleBeanName/WebServiceName/TestClient.jsp` You can use this sample application to test the web service by selecting a method, entering a value for the method, and clicking **Invoke**. The result of the method will display in the results pane.
- If you have selected to test the generated proxy using the Universal Test Client, it will be launched in a browser window at a URL similar to the following: `http://localhost:port/UTC/preload?object=BeanPackage.BeanServiceProxy`. Under Objects, expand the proxy stub to display the methods of the web service. Click the method you want to test, enter a value in the Parameters pane, and click **Invoke**. The result will be automatically generated.
- If you have selected to test the web service using the Web Services Explorer, the Explorer will open. Select the operation you want to test, enter the required information, and click **Go**. The result will display in the Status pane.

Restriction:When generating a JAX-WS web service from a class file that depends on other classes contained in separate JAR files, the project needs to reference such external jar files either from the Java EE Module Dependencies or from User Libraries. Using Java EE Module dependencies is possible when the jars are contained in the EAR project.

In the case where the jars are part of a Shared Library configured on WebSphere Application Server, then a User Library must be used to configure the project for development before generating the web service. To add a user library to the project:

1. Right-click the project and click **Properties > Java Build Path > Libraries**
2. Click **Add Library > User Library > User Libraries**
3. Create a user library, and add your utility jar to it.

Parent topic:[Creating web services and clients by using the web service wizards](#)

Related concepts:

[Java API for XML based web services](#)

[JAXB](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Web services runtime environments](#)

Related tasks:

[Generating Java beans from a JAXB Schema](#)

Creating a JAX-RS web service

Before you begin

To deploy the web service you need to have a server which has Java™ 5.0 or later JVM support defined and started. This includes:

- WebSphere® Application Server v7.0 or v8.0, selecting to install the Feature Pack for Web 2.0 AND MOBILE and the tools for WebSphere Application Server development

By default a server is created for you when you install WebSphere Application Server. This server can be seen in the Servers view.

If you choose to deploy to a server other than WebSphere Application Server, you cannot use the IBM® JAX-RS library, and must install and configure your own JAX-RS libraries through the Project Facets > JAX-RS preferences page.

For detailed information on using JAX-RS web services on WebSphere Application Server, refer to: [WebSphere Application Server information centre](#)

Parent topic: [Creating web services and clients by using the web service wizards](#)

Create a JAX-RS enabled web project

About this task

The JAX-RS web service needs to reside in a project with the JAX-RS facet enabled.

Procedure

1. In the Java EE perspective, right-click your enterprise application project and select **New > Web Project** to open the web project wizard.
2. In the **Name** field, type a name for your new web project.
3. In the Project Templates section, select the type of web template you want to use: select **Simple** to create a simple web project.
4. In the Programming Model section, select the programming model that you want to use: select **Java EE programming model**. Click **Next** to configure your new web project.
5. On the deployment page, from the list of available configuration options, click **Deployment** to open the Deployment configuration page.
 - You can change **Target runtime** by selecting another one from the drop-down box. Click **Change Features** to open the Project Facets window. On the Project Facets page, select **JAX-RS (REST Web Services)**, and click **OK**.
 - Click **Add support for WebSphere bindings and extensions** or clear this field.
 - In the **Web module version** field, select the web module version that you want to use.
 - In the **EAR membership** field, click **Add project to an EAR**, if you want to include EAR membership; clear this field if you do not want to add the web project to an EAR file.
 - In the **EAR project name** field, the name of your existent EAR file appears. You can click **Browse** to select a different EAR file.

Note: The deployment option is not available if you selected the Client-side only programming model for your new web project.
6. From the list of available configuration options, click **REST Services**.
 - A. In the **JAX-RS Implementation Library** field select the library for you server version, for example **IBM WebSphere Application Server v8.0 JAX-RS Library** or **IBM WebSphere Application Server v8.5 JAX-RS Library**. [Learn](#)

- more about libraries:** The JAX-RS libraries for each WebSphere Test Environment you have installed will be listed in this drop-down box. If you want to use a non-WebSphere library, it can be imported using the User Libraries preference page. If you select User Library, you can launch the User Libraries preference page and add a library. Once a project has been created, you can change the library by right-clicking your project and selecting **Preferences > Project Facets > JAX-RS**. You can also select to have the wizard not configure the library for you, in which case you must manually configure the classpath.
- B. If you are using a version of WebSphere Application Server earlier than v8, check **Include library with this application** and select to include it as a **Shared Library**. This adds JAX-RS jars to the classpath, as well as adding a Shared Library entry in the enhanced EAR.
 - C. **Update Deployment Descriptor** is selected by default because you will need to use a `web.xml` to configure security constraints and other behavior. A `web.xml` will be generated and updated with JAX-RS servlet information even if you have selected to not generate a deployment descriptor earlier in wizard. This option is only available when the Web Modules facet selected is v3.0 or higher.
 - D. If you have chosen to update the deployment descriptor, you will be able to change the servlet name and servlet class name, and change the URL mapping patterns.
7. You can change **Target runtime** by selecting another one from the drop-down box. Click **Change Features** to open the Project Facets window.
 - In the **Source folders on build path** field, accept the default **src** directory, or click **Add Folder**, **Edit...** or **Remove** to specify a folder for your source files.
 - In the **Default output folder:** field, specify a folder for your output files or accept the default value (`WebContent\WEB-INF\classes`).
 8. From the list of available configuration options, click **Web Module**. On the Web Module configuration page:
 - In the **Context root** field, type the name of your web project root, or accept the default (which is the name of your web project).
 - In the **Content directory** field, type the name of your content directory, or accept the default (`WebContent`).
 - Select **Generate web.xml deployment descriptor** if you want to create a deployment descriptor. You can also add a deployment descriptor to your web module later.
 9. Click **Finish**.

Create a JAX-RS web service Procedure

1. In your Web project, create a package (right-click **Java Resources > src** and select **New > Package**). Import the classes for the web service into the package.
2. Open `WebContent/WEB-INF/web.xml`. In the Design view, select the Servlet JAX-RS Servlet and click **Add** and add an Initialization parameter to the JAX-RS servlet, leaving the name and value fields empty. Save `web.xml` ignoring any errors that might be displayed.
3. In the Markers view, select the `web.xml` error about the param-name, right-click and select Quick Fix. Browse to the application and select.
4. Add the EAR containing the JAX-RS project to the server and start the server.

Results

Every web application must have a context root for the web application to deploy successfully. A context root for each Web module is defined in the application deployment descriptor during application assembly or during application deployment. The context root is combined with the defined servlet mapping from the WAR file to compose the full URL that users type to access the servlet. The context root for each deployed web application must be unique on the server. The context root can also be empty.

For instance, if a Web application used a context root of `sample/application/`, the web application request URL begins with `http://<hostname>:<port>/sample/application/`. The URL pattern of a servlet is appended to the context root of the Web application. For example, if the context root is `sample/application/` and the servlet URL mapping is `rest/api/*`, the base URI for the JAX-RS web application is `http://<hostname>:<port>/sample/application/rest/api`.

Editing a JAX-RS project

Once you have created a JAX-RS web service, you can edit it by right-clicking the project and selecting **Properties >**

Project Facets > JAX-RS. From this page you can change:

- Library type
- JAX-RS servlet name
- JAX-RS servlet class name
- URL mapping patterns

For example if you migrate your project to a different server, you can change the library provider to one supported by the new server.

Creating web service clients

You can create a web service client for a web service in your workspace or a remote service.

When you use the web services wizards to create your web service, you can create the client at the same time or you can create a client after the web service is created. For information on creating a web service client when you create the service, refer to the following web service creation tasks:

- [Creating top-down web services](#)
- [Creating bottom-up web services](#)

To create a client after you created the web service or from an imported WSDL document, refer to the following tasks:

- [Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment](#)
- [Generating a Web service client using the Axis run-time environment](#)

If you want to create a web service client for a service that is not in your workspace, you must first find the web service and import the WSDL file. For more information, refer to [Discovering and importing a web service](#).

- [Generating a web service client from a WSDL document using the IBM WebSphere JAX-WS runtime environment](#)

The web service client wizard assists you in generating a Java bean proxy and a sample application. The sample web application demonstrates how to code a proxy file.

Parent topic: [Creating web services and clients by using the web service wizards](#)

Generating a web service client from a WSDL document using the IBM® WebSphere JAX-WS runtime environment

The web service client wizard assists you in generating a Java™ bean proxy and a sample application. The sample web application demonstrates how to code a proxy file.

Before you begin

Prerequisites:

- If you are using WebSphere® Application Server, it is strongly suggested that you start the server before running the web service wizard because it may take several minutes to start the WebSphere Application Server depending on the speed of your computer. To start the server, select it in the Servers view (**Window > Show View > Servers**), right-click and select **Start**.
- Discover or import a WSDL document into the workspace. You can use a WSDL file that contains a service element. You can use a static WSDL file from your workspace, a WSDL file that resides at a URL, or if you have an annotated bean that acts as a JAX-WS web service in your workspace, the WSDL file dynamically produced by the WebSphere Application Server Feature Pack for Web Services run time. This WSDL can be found at the URL: `http://localhost:port/service_project_name/service_nameService?wsdl`. For example, in the address book JAX-WS web service that is created in the tutorial and samples that accompany the Feature Pack for Web Services the dynamic WSDL file would be found at this URL: `http://localhost:9082/jwsAddressBook/AddressBookService?wsdl`.

Restriction: If the you want to run a simple Java web service client outside of any J2EE container, you must generate the web service client into a simple Java project. The web service wizard cannot create a simple Java project. The closest form is an utility project, however running a simple Java web service client (containerless) from an utility project is not supported. You must create a simple Java project before running the web service client wizard. The JDK Compiler compliance level for the simple Java project must be set to the Java version on which the WebSphere Application Server runs.

About this task

To generate a Java client proxy and a sample application from a discovered WSDL document:

Procedure

1. Switch to the Java EE perspective (**Window > Open Perspective > Java EE**).
2. Click **File > New > Other**. Select **Web Services** to display the various web service wizards. Select the **Web Service Client** wizard. Click **Next**.
3. Web Services page: select the WSDL file that you use to generate the client. You can optionally choose to do the following:
 - A. Select the stages of web service client development that you want to complete by using the slider. This sets several default values on the remaining wizard panels.
 - Develop: develops the WSDL definition and implementation of the web service client. This includes such tasks as creating the modules, which contain the generated code, WSDL files, deployment descriptors, and Java files when appropriate.
 - Assemble: ensures the project that hosts the web service client gets associated to an EAR when required by the target application server.
 - Deploy: creates the deployment code for the client.

- Install: installs and configure the web module and EARs on the target server. If any changes to the endpoints of the WSDL file are required, they are made in this stage.
 - Start: starts the server once the client has been installed on it.
 - Test: provides various options for testing the service, such as using the Web Service Explorer or sample JSPs.
- B. Select your server: the default server is displayed. If you want to deploy your service to a different server click the link to specify a different server. This task supports the following server runtime environments:
- WebSphere Application Server V7.0 or later
 - WebSphere Application Server Liberty Profile V8.5.5
- C. Select your runtime: the default runtime is displayed. If you want to deploy your service to a different runtime click the link to specify a different runtime.
- D. Select the client project: the project that is selected in your workspace is displayed. To select a different project and EAR click the project link. You can select a web project, an EJB project, a Java project, or a Java EE application client project as the location for the client code. Ensure that the project selected as the Client Web Project is different from the Service Web Project, or the service is overwritten by the client's generated artifacts. For JAX-WS web services, the server and client projects can share the same EAR.
- E. Monitor the web service: sends the web service traffic through the TCP/IP Monitor, which enables you to watch the SOAP traffic that is generated by the web service and to test this traffic for WS-I compliance. Alternately you can manually set up a TCP/IP monitor as described in [Using the TCP/IP Monitor to test web services](#).
4. WebSphere JAX-WS Web Service Client Configuration page:
- Output folder: Select the folder where you want the client's Java classes to be generated or accept the default.
 - Target package: The web services client wizard generates a number of Java files from the specified WSDL. By default it creates a package name that is based on the namespace that is specified in the WSDL file. To override this default behavior, you can specify your own package name for the namespace in the WSDL file.
 - Generate portable client: Selecting this check box would enable you to move your web service client code from one machine to another or from one instance of WebSphere Application Server to another. If this option is selected, the WSDL document and all the XML schema and other WSDL documents that it depends upon is copied into the client project under WEB-INF/wsdl and a `file:relativeURL` pointing to this copy is then injected into the JAX-WS Service class's static initialization block.
 - Enable asynchronous invocation for generated client: If you select to enable an asynchronous client, for each method in the web service two more methods are created. These methods are polling and callback methods, which allow the client to function asynchronously.
 - Specify JAX-WS or JAXB binding files: If you created JAX-WS or JAXB custom binding files, select this check box to use them to create this web service.
 - Customize client proxy class name: You can accept the default proxy name or enter your own.
 - Generate web service deployment descriptor: For JAX-WS web services deployment information is generated dynamically by the runtime; static deployment descriptors are no longer required. Selecting this check box generates them.
 - Allow vendor extensions: If you select this check box, the `-extension` option is added to the **wsgen** or **wsimport** command. The `-extension` option specifies whether to enable or accept custom extensions for functionality not specified by the JAX-WS specification. Use of the extensions can result in applications that are not portable or do not interoperate with other implementations. For details, see [wsgen command for JAX-WS applications](#) and [wsimport command for JAX-WS applications](#) topics available in the Information Center for WebSphere Application Server.
5. Custom binding declaration page: If you selected to supply a JAX-WS or JAXB binding file on the previous page, you can browse to where they are located.
6. Proxy generation configuration page: If you selected to customize the client proxy class name, use this page to specify a proxy class name for each port.

7. Web Service Client Test page:

- Test the generated proxy: If selected, the sample client is launched in a web browser which enables you to test the function.
- Select your test facility. You can test the generated proxy in the Generic Service Client or the Web Service Explorer, or you can generate and use the sample JAX-WS 2.0 JSPs.
- Folder: If you selected to test the proxy through a JSP, you can select the folder where the JSP is located, and you can select the methods that are included in the JSP.
- Methods: Select the methods to expose. If you selected to enable asynchronous invocation, the asynchronous methods should be listed as well.
- Run test on server: starts the server for you automatically.

8. Click **Finish**. If you selected to test the proxy, the test client opens in a browser window.

Results

The generated Java bean proxy provides a remote procedure call interface to the web service. The sample Web application demonstrates how to code the proxy file.

Once you have generated your Java client proxy, you may test the methods of the web service through the proxy using web services sample JSPs, or the Web Services Explorer.

- If you selected to test the web service by using the Web Services Explorer, the Explorer opens. Select the operation that you want to test, enter the required information, and click **Go**. The result displays in the Status pane.
- If you selected to test a Java thin client, the system output of running your Java thin client is shown in the Console view. You can run the Java thin client by clicking **Run** in the **Run Configurations** dialog box. As a plain Java application, the client uses a corresponding WebSphere Application Server thin client runtime container JAR.

Parent topic:[Creating web service clients](#)

Related concepts:

[Java API for XML based web services](#)

[JAXB](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Web services runtime environments](#)

Creating web services by using annotations

With the Java™ API for XML-Based web services, you can use annotations in your Java code to simplify creating web services.

- [Annotations for creating web services overview](#)

You can use annotations in your Java code to create web services. Annotations specify metadata that is associated with web service implementations.

- [Annotating a Java bean to create a web service](#)

You can annotate types, methods, fields, and parameters in your Java bean to specify a web service.

- [Annotating an EJB bean to create a web service](#)

You can annotate types, methods, fields, and parameters in your EJB bean to specify a web service.

- [Creating a web service from a Java bean and a WSDL file](#)

You can use annotations to associate an implementation Java bean with an existing WSDL service contract.

- [Creating a Web service from an EJB bean and a WSDL file](#)

You can use annotations to associate an implementation EJB bean with an existing WSDL service contract.

- [Validation of web services annotations](#)

When you are using annotations with Java code, the product validates the usage and values of the annotations. Through this validation, you can detect problems before deploying your web service.

- [JAX-WS annotations reference](#)

You can use annotations with web services to configure bindings and handler chains; and to set names of portType, service, and other WSDL parameters. Annotations are used at build time to map Java to WSDL files and XML schema, and at run time to control how the JAX-WS runtime environment processes and responds to web service invocations.

Parent topic: [Developing Web services and clients](#)

Annotations for creating web services overview

You can use annotations in your Java™ code to create web services. Annotations specify metadata that is associated with web service implementations.

The Java API for XML-Based Web Services (JAX-WS) programming standard relies on annotations to simplify the development of web services. Annotations describe two aspects of web services: how a server-side service implementation is accessed as a web service, and how a client-side Java class accesses web services.

The JAX-WS programming standard supports annotating Java classes with metadata that is used to define a service endpoint application as a web service and to specify how a client can access the service. The JAX-WS standard supports the use of annotations that are based on several Java Specification Requests (JSRs):

- A Metadata Facility for the Java Programming Language (JSR 175),
- Web Services Metadata for the Java Platform (JSR 181)
- Java API for XML-Based web services (JAX-WS) 2.0 (JSR 224).
- Common Annotations for the Java Platform (JSR 250)

Using annotations from the JSR 181 standard, you can annotate a service implementation class or a service interface.

Then, you can generate a web service with a wizard or by publishing the application to a server. Using annotations within both Java source code and Java classes simplifies web service development. If you use annotations in this way it defines more information that is typically obtained from deployment descriptor files, Web Services Description Language (WSDL) files, or mapping metadata from XML and WSDL into source artifacts.

You can use annotations to configure bindings and handler chains; and to set names of portType, service, and other WSDL parameters. You can also use annotations at build-time to map Java to WSDL and schema, and at run time to control how the JAX-WS runtime processes and responds to web service invocations.

The product uses annotations in clients that you create through the web service client wizard. The wizard generates a client proxy and a static service class with annotations. These annotations specify how the client accesses the web service.

For detailed information about annotations that you can use, see the related reference.

Supported annotation-based web service scenarios

The following web service scenarios are supported by the tools and WebSphere® runtime environments:

Table 1. Supported runtime environments. Run times supported by annotation-based web service scenarios.

Scenario	Runtime environment	Comments
----------	---------------------	----------

Bottom-up Java bean	WebSphere Application Server v7.0 and later	<p>Can be done by using annotations alone or in combination with the web service wizards. If the bean already has a <code>@javax.jws.WebService</code> annotation, many of the fields in the wizard is disabled because the wizard does not need to generate a delegate bean for you. You are only able to select to generate a WSDL file. If you added only the <code>@javax.jws.WebService</code> to your Java bean and want to enable other options such as SOAP 1.2 binding or MTOM, exit the wizard and either remove the annotation or proceed to create the web service by using the annotations documentation. The wizard does not enable you to append new annotations to a pre-existing partially annotated bean.</p>
Bottom-up EJB 2.x	None	Not supported by using annotations
Bottom-up EJB 3.0	WebSphere Application Server v7.0 and later	Cannot use a mixture of annotations and the web services wizard. You must use annotations and deploy the EJB bean to the application server.
Meet-in-the-middle Java bean	WebSphere Application Server v7.0 and later	<p>This scenario enables you to map a web service annotated Java bean to a WSDL document using the <code>wsdlLocation</code> attribute. Note: If you want to use SOAP 1.2:</p> <pre>@javax.xml.ws.BindingType (value=javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)</pre> <p>you must specify the <code>wsdlLocation</code> attribute of the <code>@WebService</code> annotation, such as in the following example:</p> <pre>@javax.jws.WebService (targetNamespace="http://p/", ..., wsdlLocation="WEB-INF/wsdl/EchoService.wsdl")</pre> <p>You cannot use the WSDL file dynamically generated by WebSphere Application Server but must have a WSDL file created beforehand. Alternatively, you can use the web services wizards, which generates a WSDL file for you if you select to use SOAP 1.2 when you generate a web service from a Java bean. For more information, see: Creating a web service from a Java bean and a WSDL file.</p>

Meet-in-the-middle EJB 3.0	WebSphere Application Server v7.0 and later	This scenario enables you to map a web service annotated EJB bean to a WSDL document by using the wsdlLocation attribute. For more information, see: Creating a Web service from an EJB bean and a WSDL file .
----------------------------	---	--

Example

For example, you can embed a `@WebService` tag in the Java source to expose a bean as a web service. `@WebService`

```
public class QuoteBean implements StockQuote {  
  
    public float getQuote(String sym) { ... }  
  
}
```

The annotation `@WebService` tells the server runtime environment to expose all public methods on that bean as a web service. You can control more levels of granularity by adding other annotations on individual methods or parameters. Using annotations makes it much easier to expose Java artifacts as web services. In addition, as you create artifacts by using some of the top-down mapping tools that start from a WSDL file, annotations are included within the source and Java classes as a way of capturing the metadata along with the source files.

Parent topic: [Creating web services by using annotations](#)

Related concepts:

[Java API for XML based web services](#)

[JAXB](#)

[IBM WebSphere JAX-WS runtime environment](#)

[Web services runtime environments](#)

Annotating a Java bean to create a web service

You can annotate types, methods, fields, and parameters in your Java™ bean to specify a web service.

Before you begin

Prerequisites:

- Your workspace contains a Java bean that has at least one public method.
- This Java bean is in a web project or basic Java project.

About this task

To annotate your Java bean:

Procedure

1. In the Enterprise Explorer view, double-click your Java bean to open the file in the Java editor.
2. On the Java class that implements your web service, specify the `@WebService` annotation. Also specify the attributes, if any, that you want the annotation to have.
 - Most errors that display when adding annotations can be resolved using the suggested quickfixes. To display the quickfixes, click the error marker.
 - To add annotations or attributes to existing annotations, you can use the Annotations view rather than manually adding this information to the class. This view provides basic validation and guidance when working with annotations.
3. Optional: Use these and other annotations to customize your web service further:
 - On the Java class that implements your web service, specify the `@WebMethod` annotation on each method that you want to customize for the service. You can use this annotation to exclude a method from your service. By default, all public methods are exposed in a service, including inherited methods that are under the Object class.
 - On the methods that are exposed in your web service, use the `@WebParam` and `@WebResult` annotations to customize the mapping of your parameters and results to message parts and XML elements.
 - On an exception class, specify the `@WebFault` annotation to map your class to a WSDL fault.

Results

Once the annotations have been added to the bean, the Services view should list the web service under the JAX-WS heading. From this view you can test the web service by right-clicking it and selecting **Test with Web Services Explorer** or **Test with Generic Service Client**. You can also generate deployment descriptors and manage the policy sets associated with the service from this view.

- [Creating a web service from an annotated Java bean by using a wizard](#)

After annotating a Java bean, you can generate a web service application by using the web services wizard. With the wizard, you can create a WSDL file in your web services project before you deploy the application to a server.

- [Creating a web service from an annotated Java bean by publishing to a server](#)

After annotating a Java bean, you can generate a web service application by publishing the application project of the bean directly to a server. When your web service is generated, no WSDL file is created in your project.

Parent topic: [Creating web services by using annotations](#)

Creating a web service from an annotated Java bean by using a wizard

After annotating a Java™ bean, you can generate a web service application by using the web services wizard. With the wizard, you can create a WSDL file in your web services project before you deploy the application to a server.

Before you begin

Prerequisite: For a Java bean in your workspace, you must have specified web services annotations, including at least the `@WebService` annotation.

About this task

Note: If the bean already has a `@javax.jws.WebService` annotation, many of the fields in the wizard will be disabled because the wizard does not need to generate a delegate bean for you. You will only be able to select to generate a WSDL file. If you have added only the `@javax.jws.WebService` to your Java bean and want to enable other options such as SOAP 1.2 binding or MTOM, you should exit the wizard and either remove the annotation or proceed to create the web service using the annotations documentation. The wizard will not allow you to append new annotations to a pre-existing partially annotated bean.

To create a web service from an annotated Java bean:

Procedure

1. In the menu bar, click **File > New > Other**.
2. In the New window, click **Web Services > Web Service**. Click **Next**.
3. In the web services wizard, select Bottom up Java bean web service in the **Web service type** list.
4. In the **Service implementation** field, type the name of the Java bean that implements your web service, or use the **Browse** button to select this bean.
5. Under **Configuration**:
 - A. Verify that the server is IBM® WebSphere® Application Server version 7.0 or up. If not, click **Server** to select it.
 - B. Verify that the web service runtime is the IBM WebSphere JAX-WS runtime. If not, click **Web service runtime** to select it.
 - C. Click **Service project** to select the project that implements your web service.
 - D. Click **Service EAR project** to select the application project that contains your web service.
6. Click **Next**.
7. If you want to create a Web Services Description Language (WSDL) service contract from your Java bean, select the **Generate WSDL into the project** check box. A WSDL file for your web service will be created in the `WebContent\WEB-INF\wsdl` folder of the project that implements your web service.
8. Click **Finish** to complete the wizard

Results

Your Java bean is published as a web service on the server that you specified. If the level of service generation that you selected in the wizard was Start or Test, your web service is also started. **Note:** If you used the `@BindingType` annotation in your Java bean to specify a WSDL 1.1 SOAP 1.2 binding, the wizard generates a WSDL file with this type of binding.

For more information about the web services wizard, including other options for your web service, see the related tasks.

Parent topic: [Annotating a Java bean to create a web service](#)

Creating a web service from an annotated Java bean by publishing to a server

After annotating a Java™ bean, you can generate a web service application by publishing the application project of the bean directly to a server. When your web service is generated, no WSDL file is created in your project.

Before you begin

- For a Java bean in your workspace, you have already specified web services annotations, including at least the `@WebService` annotation. If there are validation errors or warnings in the Java editor, you should address these to prevent potential problems at run time.
- You have already created a server in your workspace.

About this task

To create a web service from an annotated Java bean:

Procedure

1. Open the Servers view, if it is not already open, by clicking **Window > Show View > Servers** in the menu bar.
2. If the application project that contains your Java bean has not been published to a server:
 - A. In the Servers view, right-click the server where you want to publish your application.
 - B. In the menu, click **Add and Remove Projects**.
 - C. In the Add and Remove Projects window, from the **Available projects** list, select the application project that contains your Java bean.
 - D. Click **Add**; then click **Finish**.
3. If you haven't specified a preference for automatically publishing to local servers, right-click the server where you want to publish your application and click **Publish**.

Results

Your Java bean is published as a web service on the server. JAXB classes are generated and packaged into the application at the point of publication. **Note:** If you want to use SOAP 1.2:`@javax.xml.ws.BindingType`

```
(value=javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
```

you must specify the `wSDLLocation` attribute of the `@WebService` annotation, such as in the following example:

```
@javax.jws.WebService (targetNamespace="http://p/", ..., wSDLLocation="WEB-INF/wSDL/EchoService.wSDL")
```

This means that you cannot use the WSDL file dynamically generated by WebSphere® Application Server but must have a WSDL file created beforehand and follow the instructions in [Creating a web service from a Java bean and a WSDL file](#).

Alternatively you can use the web services wizards, which will generate a WSDL file for you if you select to use SOAP 1.2 when generating a web service from a Java bean.

Parent topic: [Annotating a Java bean to create a web service](#)

Annotating an EJB bean to create a web service

You can annotate types, methods, fields, and parameters in your EJB bean to specify a web service.

Before you begin

Prerequisites:

- Your workspace contains an EJB 3.x Session bean that has at least one public method.
- This EJB bean is in a JAX-WS enabled EJB project.
- The project's Target Runtime is WebSphere® Application Server v7.0 or higher.

About this task

The steps for creating a web service from an EJB bean using annotations is as follows:

1. Annotate your EJB bean with the `@WebService` annotation, and any other annotations required for your implementation.
2. Create JMS or HTTP router modules for the web service as described in: [Creating web service router modules](#). New in WebSphere Application Server v8, if you package your EJB application in a WAR module, you do not need to create router modules.
3. Publish the application to a server as described in: [Creating a web service from an annotated EJB bean by publishing to a server](#)

To annotate your EJB bean:

Procedure

1. In the Enterprise Explorer view, double-click your Java™ bean to open the file in the Java editor.
2. On the Java class that implements your web service, specify the `@WebService` annotation, as well as a `@Stateless`, `@Stateful` or `@Singleton` annotation. Also specify the attributes, if any, that you want the annotation to have.
 - Most errors that display when adding annotations can be resolved using the suggested quickfixes. To display the quickfixes, click the error marker.
 - To add annotations or attributes to existing annotations, you can use the Annotations view rather than manually adding this information to the class. This view provides basic validation and guidance when working with annotations.
3. Optional: Use these and other annotations to customize your web service further:
 - On the Java class that implements your web service, specify the `@WebMethod` annotation on each method that you want to customize for the service. You can use this annotation to exclude a method from your service. By default, all public methods are exposed in a service, including inherited methods that are under the Object class.
 - On the methods that are exposed in your web service, use the `@WebParam` and `@WebResult` annotations to customize the mapping of your parameters and results to message parts and XML elements.
 - On an exception class, specify the `@WebFault` annotation to map your class to a WSDL fault.

Results

Once the annotations have been added to the bean, the Services view should list the web service under the JAX-WS heading. From this view you can test the web service by right-clicking it and selecting **Test with Web Services Explorer** or **Test with Generic Service Client**. You can also generate deployment descriptors and manage the policy sets associated with the service from this view.

- [Creating web service router modules](#)

The Create Router Modules wizard enables a set of web services within an Enterprise Application Archive (EAR) file. For each web service-enabled EJB JAR file in the EAR file, it adds an HTTP router, a JMS router, or both to the EAR. Each

router module provides a web service endpoint for a particular transport. For example, an HTTP router module can be added so that the web service can receive requests over the HTTP transport, and a JMS router module can be added so that the web service can receive requests from a JMS queue or topic. The Create Router Modules wizard was formerly known as the Endpoint Enabler.

- **Creating a web service from an annotated EJB bean by publishing to a server**

After annotating an EJB bean, you can generate a web service application by publishing the application project of the bean directly to a server. When your web service is generated, no WSDL file is created in your project.

Parent topic: [Creating web services by using annotations](#)

Creating web service router modules

The Create Router Modules wizard enables a set of web services within an Enterprise Application Archive (EAR) file. For each web service-enabled EJB JAR file in the EAR file, it adds an HTTP router, a JMS router, or both to the EAR. Each router module provides a web service endpoint for a particular transport. For example, an HTTP router module can be added so that the web service can receive requests over the HTTP transport, and a JMS router module can be added so that the web service can receive requests from a JMS queue or topic. The Create Router Modules wizard was formerly known as the Endpoint Enabler.

Before you begin

Important: Applicable edition: Full profile

To use the Create Router Modules wizard, you must have an EAR project that contains a web service-enabled EJB project. If your EJB application is being deployed in a WAR module to WebSphere® Application Server V8, you do not need to create router modules.

About this task

The Create Router Modules wizard creates JAX-WS JMS Listener message-driven beans. If there is at least one JAX-WS web service present, a JAX-WS router module is generated: `com.ibm.ws.websvcs.transport.jms.JMSListenerMDB`

Note: The router module is not required when you deploy to Liberty.

Procedure

1. Select one of the following elements:
 - The web service object under the Services folder in the Enterprise Explorer or in the Services view
 - The `application.xml` file or EAR file within an EAR project
 - An EJB object
 - A `webservices.xml` file
 - An `ejb-jar.xml` file
2. Right-click the selected element, and select **Web Services > Create Router Modules**.
3. Select which type of transports you want to use: HTTP, JMS, or both. Enter the router module names and any other information that is required to enable the type of transport you want to use, or accept the defaults. Click **OK**.

Results

After you run this wizard, a web project for the HTTP router or an EJB project for the JMS router is created, depending on which transports were used. **Remember:** Do not modify the contents of the EJB module or the web module that was generated by using the Create Router Modules wizard. If you do, an error occurs during run time. An error such as the following message is displayed: `Error - WWS3142E: Error: Could not find Web services engine.]: javax.servlet.ServletException: WWS3142E: Error: Could not find Web services engine.`

Parent topic: [Annotating an EJB bean to create a web service](#)

Parent topic: [Deploying web services](#)

Creating a web service from an annotated EJB bean by publishing to a server

After annotating an EJB bean, you can generate a web service application by publishing the application project of the bean directly to a server. When your web service is generated, no WSDL file is created in your project.

Before you begin

Prerequisites:

- For an EJB bean in your workspace, you have already specified web services annotations, including at least the `@WebService` and `@Stateless` annotations. If there are validation errors or warnings in the Java™ editor, you should address these to prevent potential problems at run time.
- You have created the router modules for the web service.
- You have already created a server in your workspace.

About this task

To create a web service from an annotated EJB bean:

Procedure

1. Open the Servers view, if it is not already open, by clicking **Window > Show View > Servers** in the menu bar.
2. If the application project that contains your EJB bean has not been published to a server:
 - A. In the Servers view, right-click the server where you want to publish your application.
 - B. In the menu, click **Add and Remove Projects**.
 - C. In the Add and Remove Projects window, from the **Available projects** list, select the application project that contains your EJB bean.
 - D. Click **Add**; then click **Finish**.
3. If you haven't specified a preference for automatically publishing to local servers, right-click the server where you want to publish your application and click **Publish**.

Results

Your EJB bean is published as a web service on the server. JAXB classes are generated and packaged into the application at the point of publication.

Parent topic: [Annotating an EJB bean to create a web service](#)

Creating a web service from a Java bean and a WSDL file

You can use annotations to associate an implementation Java™ bean with an existing WSDL service contract.

Before you begin

- Your workspace must contain a Java bean that has been annotated as a web service.
- You must have an existing Web Services Description Language (WSDL) service contract.

About this task

To create a web service:

Procedure

1. In the Enterprise Explorer view, double-click your Java bean to open the file in the Java editor.
2. In the editor or Annotations view, specify a value for the wsdlLocation attribute of the @WebService annotation. This value should be the Web address of the WSDL service contract that you want to associate with your bean.
3. Optional: Use annotations to associate other Java elements with WSDL elements. For example, you might associate a Java method name with a WSDL operation name, or a Java parameter with a WSDL schema element. For detailed information about these and other annotations, see the related reference.
4. Save your Java file.

What to do next

Postrequisite: Now you can [Create a web service from your annotated Java bean by publishing it to a server.](#)

Note: As you use annotations to associate your Java bean with a WSDL service contract, the JAX-WS annotations processor ensures that your Java code is consistent with the WSDL contract as required by the JAX-WS and JSR-181 standards. If there are inconsistencies, you will see warnings or errors in the code, along with suggestions for correction.

Parent topic: [Creating web services by using annotations](#)

Creating a Web service from an EJB bean and a WSDL file

You can use annotations to associate an implementation EJB bean with an existing WSDL service contract.

Before you begin

Prerequisites:

- Your workspace must contain an EJB 3.0 bean that has been annotated as a Web service (it must contain the `@Stateless` annotation).
- You must have an existing Web Services Description Language (WSDL) service contract.

About this task

To create a Web service:

Procedure

1. In the Enterprise Explorer view, double-click your EJB bean to open the file in the Java™ editor.
2. In the editor or Annotations view, specify a value for the `wsdlLocation` attribute of the `@WebService` annotation. This value should be the Web address of the WSDL service contract that you want to associate with your bean.
3. Optional: Use annotations to associate other Java elements with WSDL elements. For example, you might associate a Java method name with a WSDL operation name, or a Java parameter with a WSDL schema element. For detailed information about these and other annotations, see the related reference.
4. Save your EJB file.

What to do next

Postrequisite: Now you can [Create a Web service from your annotated Enterprise bean by publishing it to a server.](#)

Note: As you use annotations to associate your EJB bean with a WSDL service contract, the JAX-WS annotations processor ensures that your Java code is consistent with the WSDL contract as required by the JAX-WS and JSR-181 standards. If there are inconsistencies, you will see warnings or errors in the code, along with suggestions for correction.

Parent topic: [Creating web services by using annotations](#)

Validation of web services annotations

When you are using annotations with Java™ code, the product validates the usage and values of the annotations. Through this validation, you can detect problems before deploying your web service.

When developing web services, you can benefit from two levels of validation. The first level involves validating syntax and Java-based default values. This level of validation is performed by the Eclipse Java Development Tools (JDT). The second level of validation involves implicit default checking, as well as verification of Web Services Description Language (WSDL) contracts. This second level is performed by a JAX-WS annotations processor.

When you select the WebSphere® Web 7.0, 8.0, or 8.5 facet for a project, you enable the JAX-WS annotations processor. (This processor extends the Eclipse Annotations Processing Tool (APT) framework.) When you enable the annotations processor, warnings and errors for annotations are displayed like Java errors. You can work with these warnings and errors in various workbench locations, such as the Problems view.

For example, you can use the `wsdlLocation` of the `@WebService` annotation to enforce consistency between your WSDL contracts, Java Service Endpoint Interfaces (SEIs), and implementation beans. By using the annotations processor to detect problems at build time, you can prevent these problems from occurring at run time.

While you are annotating Java code for a web service, your WSDL file might sometimes be slow or impossible to access from the workspace. In this case, you can disable verification of WSDL contracts in your project:

1. Right-click your web service project in the Enterprise Explorer; then click **Properties** in the menu.
2. In the Properties window, click **Java Compiler > Annotations Processing**.
3. Under **Processor options**, select the key `com.ibm.ws.ast.jws.annotations.processor.validateWSDL`.
4. Click the **Edit** button.
5. In the Edit Processor Option window, type `off` in the **Value** field.
6. Click **OK** twice.

Parent topic: [Creating web services by using annotations](#)

JAX-WS annotations reference

You can use annotations with web services to configure bindings and handler chains; and to set names of portType, service, and other WSDL parameters. Annotations are used at build time to map Java™ to WSDL files and XML schema, and at run time to control how the JAX-WS runtime environment processes and responds to web service invocations.

Purpose

You can apply annotations to these Java objects:

- Types such as a class, enum, or interface
- Methods
- Fields that represent local instance variables in a class
- Parameters in a method

- **Web Services metadata annotations (JSR 181)**

Using annotations from the JSR 181 specification, you can annotate a web service implementation class or a web service interface. Then, you can generate a web service with a wizard or by publishing the application to a server.

- **JAX-WS 2.0 Annotations (JSR 224)**

The JSR 224 specification defines annotations for JAX-WS 2.0.

- **JAX-WS Common Annotations (JSR 250)**

The JSR 250 specification includes annotations for injecting a resource into an endpoint implementation class, and for managing application lifecycle.

- **Rules for methods in classes annotated with @WebService**

When you use the @WebService annotation, several rules control how methods are exposed in your web service and how you can use the @WebMethod annotation.

Parent topic: [Creating web services by using annotations](#)

Web Services metadata annotations (JSR 181)

Using annotations from the JSR 181 specification, you can annotate a web service implementation class or a web service interface. Then, you can generate a web service with a wizard or by publishing the application to a server.

Note: The Java™ class that contains each annotation in the JSR 181 standard is named `javax.jws.xxx`, where `xxx` is the name of the annotation after the '@' character. For example, the Java class name that corresponds to `@WebService` is `javax.jws.webservice`.

Name:	Description:	Properties:	Definitions:
-------	--------------	-------------	--------------

<p>@WebService</p>	<p>This annotation marks either a Java class or a service endpoint interface (SEI) as implementing a web service interface. Three of the properties of the annotation can be used only with a service endpoint implementation class for a JavaBeans endpoint. These properties are <code>serviceName</code>, <code>portName</code>, and <code>endpointInterface</code>. If the annotation references an SEI through the <code>endpointInterface</code> attribute, then the SEI must also have the <code>@WebService</code> annotation. By default, all public methods in a class that specifies the <code>@WebService</code> annotation are exposed in the web service.</p>	<p>Annotation target: <code>TypeProperties</code>:name The name of the <code>wsdl:portType</code>. The name property maps to an SEI or a Java implementation class. The default value is the unqualified name of the Java interface or class. (String)targetNamespace Specifies the XML namespace of the WSDL and XML elements that are generated from the web service. If the <code>@WebService.targetNamespace</code> annotation is on a service endpoint interface. The <code>targetNamespace</code> is used for the namespace for the <code>wsdl:portType</code> (and associated XML elements). If the <code>@WebService.targetNamespace</code> annotation is on a service implementation bean that does not reference a service endpoint interface (through the <code>endpointInterface</code> annotation element), the <code>targetNamespace</code> is used for both the <code>wsdl:portType</code> and the <code>wsdl:service</code> (and associated XML elements). If the <code>@WebService.targetNamespace</code> annotation is on a service implementation bean that does reference a service endpoint interface (through the <code>endpointInterface</code> annotation element), the <code>targetNamespace</code> is used for only the <code>wsdl:service</code> (and associated XML elements). The default value is the namespace that is mapped from the package name that contains the web service. (String)serviceName Specifies the service name of the web service:</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({TYPE}) public @interface WebService{ String name() default ""; String targetNamespace() default ""; String serviceName() default ""; String wsdlLocation() default ""; String endpointInterface() default ""; String portName() default ""; };</pre>
--------------------	---	---	--

		<p>web service: <code>wsdl:service</code>. This property is not allowed on endpoint interfaces. The default value is the simple name of the Java class + <i>Service</i>. (String)endpointInterface Specifies the qualified name of the service endpoint interface that defines the services' abstract web service contract. If specified, the service endpoint interface is used to determine the abstract WSDL contract. (String)portName The <code>wsdl:portName</code>. The <code>portName</code> property is the name of the endpoint port. The default value is <code>WebService.name + Port</code>. (String)wsdlLocation Specifies the web address of the WSDL document that defines the web service. The web address is either relative or absolute. (String)</p>	
--	--	--	--

<p>@WebMethod</p>	<p>This annotation denotes a method that is a web service operation. You can apply this annotation to public methods in a client or server Service Endpoint Interface (SEI), or in a service endpoint implementation class for a JavaBeans endpoint. The @WebMethod annotation is only supported on classes that have the @WebService annotation.</p>	<p>Annotation target: MethodProperties: operationName Specifies the name of the <code>wsdl:operation</code> that matches this method. The default value is the name of the Java method. (String)action Defines the action for this operation. For SOAP bindings, this value determines the value of the SOAPAction header. (String)exclude Specifies whether to exclude a method from the web service. This property can be used only with an implementation class. Used to stop an inherited method from being exposed as part of this web service. If this element is specified, other elements must not be specified for the @WebMethod. This property is not allowed on endpoint interfaces. The default value is <code>false</code>. (Boolean)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({TYPE}) public @interface WebMethod{ String operationName() default ""; String action() default ""; boolean exclude() default false; };</pre>
<p>@Oneway</p>	<p>This annotation denotes a method as a one-way operation for a web service. It has an input message but no output message. This annotation can be used only on methods that have no return value. You can apply this annotation to public methods in a client or server Service Endpoint Interface (SEI), or in a service endpoint implementation class for a JavaBeans endpoint.</p>	<p>Annotation target: MethodThere are no properties on the OneWay annotation.</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({TYPE}) public @interface OneWay{ };</pre>

<p>@WebParam</p>	<p>This annotation customizes the mapping of an individual parameter to a web service message part and XML element. You can apply this annotation to public methods in a client or server Service Endpoint Interface (SEI), or in a service endpoint implementation class for a JavaBeans endpoint.</p>	<p>Annotation target: ParameterProperties: name if the operation is remote procedure call (RPC) style and the partName attribute is not specified, then this is the name of the wsdl:part attribute that represents the parameter. If the operation is document style or the parameter maps to a header, then name is the local name of the XML element that represents the parameter. This attribute is required if the operation is document style, the parameter style is BARE, and the mode is OUT or INOUT. If the partName attribute is specified, then the name attribute is ignored. (String)partName Defines the name of wsdl:part attribute that represents this parameter. Use this only if the operation is RPC style, or the operation is document style and the parameter style is BARE. (String)targetNamespace Specifies the XML namespace of the XML element for the parameter. It applies only for document bindings when the attribute maps to an XML element. The default value is the targetNamespace for the web service. (String)mode The value represents the direction that the parameter flows for this method. Valid values are IN, INOUT, and OUT. (String)header Specifies whether the parameter is in a message header rather than a message body. The default value is false. (Boolean)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({PARAMETER}) public @interface WebParam{ public enum Mode{ IN, OUT, INOUT }; String name() default ""; String partName() default ""; String targetNamespace() default ""; Mode mode() default Mode.IN boolean header() default false; };</pre>
------------------	---	---	---

<p>@WebResult</p>	<p>This annotation customizes how a return value maps to a WSDL part or XML element. You can apply this annotation to public methods in a client or server Service Endpoint Interface (SEI), or in a service endpoint implementation class for a JavaBeans endpoint.</p>	<p>Annotation target: MethodProperties:</p> <p>name Specifies the name of the return value as it is listed in the WSDL file and found in messages on the wire. For RPC bindings, this is the name of the <code>wsdl:part</code> attribute that represents the return value. For document bindings, the <code>name</code> parameter is the local name of the XML element that represents the return value. The default value is <code>return</code> for RPC and DOCUMENT/WRAPPE D bindings. The default value is the method <code>name + Response</code> for DOCUMENT/BARE bindings.</p> <p>(String)targetNamespace Specifies the XML namespace for the return value. Use this annotation only for document bindings when the return value maps to an XML element. The default value is the <code>targetNamespace</code> for the web service.</p> <p>(String)header Specifies whether the result is carried in a header. The default value is <code>false</code>.</p> <p>(String)partName Specifies the part name for the result with RPC or DOCUMENT/BARE operations. The default value is <code>@WebResult.name</code>.</p> <p>(Boolean)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({METHOD}) public @interface WebResult{ String name() default "return"; String targetNamespace default() ""; boolean header() default false; String partName() default ""; };</pre>
-------------------	--	---	---

<p>@HandlerChain</p>	<p>This annotation associates the web service to an externally defined handler chain. You can use this annotation if the handler configuration must be shared across multiple web services, and embedding handler configuration in the application source is inefficient. Apply this annotation to a client or server Service Endpoint Interface (SEI), or to a service endpoint implementation class for a JavaBeans endpoint.</p>	<p>Annotation target: TypeProperties: file Specifies the location of the handler chain file. The file location is either an absolute java.net.URL in external form or a relative path from the source or class file. (String) name Specifies the name of the handler chain in the configuration file. (String)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({TYPE}) public @interface HandlerChain{ String file(); String name(); };</pre>
<p>@SOAPBinding</p>	<p>This annotation specifies the mapping of the web service onto the SOAP message protocol. You can apply this annotation to a type or public methods in a client or server Service Endpoint Interface (SEI), or in a service endpoint implementation class for a Java beans endpoint. The method-level annotation is limited by what it can specify. Use the annotation only if the <code>style</code> property has the value <code>DOCUMENT</code>. If the method-level annotation is not specified, the @SOAPBinding behavior from the type is used.</p>	<p>Annotation target: Type or MethodProperties: style Defines encoding style for messages that are sent to and from the web service. The valid values are <code>DOCUMENT</code> and <code>RPC</code>. The default value is <code>DOCUMENT</code>. (String) use Defines the encoding that is used for messages that are sent to and from the web service. The default value is <code>LITERAL</code>. <code>ENCODED</code> is not a supported value. (String) parameterStyle Determines whether the method's parameters represent the entire message body, or if the parameters are elements that are wrapped inside a top-level element that is named after the operation. Valid values are <code>WRAPPED</code> or <code>BARE</code>. You can use the <code>BARE</code> value with <code>DOCUMENT</code> style bindings. The default value is <code>WRAPPED</code>. (String)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({TYPE}) public @interface SOAPBinding{ public enum Style{ DOCUMENT, RPC }; public enum Use{ LITERAL, }; public enum ParameterStyle{ BARE, WRAPPED }; Style style() default Style.DOCUMENT Use use() default Use.LITERAL ParameterStyle parameterStyle() default ParameterStyle.WRAPPED; };</pre>

JAX-WS 2.0 Annotations (JSR 224)

The JSR 224 specification defines annotations for JAX-WS 2.0.

Note: The Java™ class that contains each annotation in the JSR 224 standard is named `javax.xml.ws.xxx`, where `xxx` is the name of the annotation after the '@' character. For example, the Java class name for the `@BindingType` annotation is `javax.xml.ws.bindingtype`.

Name:	Description:	Properties:	Definitions:
<p><code>@BindingType</code></p>	<p>This annotation specifies which binding to use when you publish an endpoint of this type. If the annotation is not specified, the default value is <code>SOAP11_HTTP_BINDING</code>. You can apply this annotation to a JavaBeans implementation class for a service endpoint that is based on the Service Endpoint Interface or the Provider interface. Important: Use the <code>@BindingType</code> annotation on the JavaBeans endpoint implementation class to enable Message Transmission Optimization Mechanism (MTOM), by specifying either <code>SOAP11_HTTP_MTOM_BINDING</code> or <code>SOAP12_HTTP_MTOM_BINDING</code> as the value for the annotation.</p>	<p>Annotation target: TypeProperties: - value Indicates the binding identifier web address. (String)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({TYPE}) public @interface BindingType{ String value() default SOAP11_HTTP_BINDING; };</pre>

<p>@RequestWrapper</p>	<p>This annotation supplies the JAXB generated request wrapper bean, the element name, and the namespace for serialization and deserialization with the request wrapper bean that is used at runtime. When you start with a Java object, this element is used to resolve overloading conflicts in document literal mode. Only the <code>className</code> attribute is required in this case. This annotation can be applied to methods in a client or server Service Endpoint Interface (SEI), or in a service endpoint implementation class for a JavaBeans endpoint.</p>	<p>Annotation target: MethodProperties: - localName Specifies the local name of the XML schema element that represents the request wrapper. The default value is the <code>operationName</code> as defined in <code>javax.jws.WebMethod</code> annotation. (String)- targetNamespace Specifies the XML namespace of the request wrapper method. The default value is the target namespace of the SEI. (String)- className Specifies the name of the class that represents the request wrapper. (String)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({METHOD}) public @interface RequestWrapper{ String localName() default ""; String targetNamespace() default ""; String className() default ""; };</pre>
<p>@ResponseWrapper</p>	<p>This annotation supplies the JAXB-generated response wrapper bean, the element name, and the namespace for serialization and deserialization with the response wrapper bean that is used at run time. When you start with a Java object, use this element to resolve overloading conflicts while in document literal mode. Only the <code>className</code> attribute is required in this case. You can apply this annotation to methods in a client or server Service Endpoint Interface (SEI), or in a service endpoint implementation class for a Java beans endpoint.</p>	<p>Annotation target: MethodProperties: - localName Specifies the local name of the XML schema element that represents the request wrapper. The default value is the <code>operationName</code> <i>+response</i>. <code>operationName</code> is defined in <code>javax.jws.WebMethod</code> annotation. (String)- targetNamespace Specifies the XML namespace of the request wrapper method. The default value is the target namespace of the SEI. (String)- className Specifies the name of the class that represents the response wrapper. (String)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({METHOD}) public @interface ResponseWrapper{ String localName() default ""; String targetNamespace() default ""; String className() default ""; };</pre>

<p>@ServiceMode</p>	<p>This annotation specifies whether a service provider must have access to an entire message protocol or just to the message payload. Important: The @ServiceMode annotation is only supported on classes that have the @WebServiceProvider annotation.</p>	<p>Annotation target: TypeProperties: - value Indicates whether the provider class accepts the payload of the message, PAYLOAD, or the entire message MESSAGE. The default value is PAYLOAD. (String)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({TYPE}) @Inherited public @interface ServiceMode{ Service.Mode value() default javax.xml.ws.Service.Mode.PAY LOAD; };</pre>
<p>@WebFault</p>	<p>This annotation maps WSDL faults to Java exceptions. Use this annotation to capture the name of the fault; this capturing occurs during serialization of the JAXB type that is generated from a global element that is referred to by a WSDL fault message. You can also use this annotation to customize the mapping of service-specific exceptions to WSDL faults. You can apply this annotation to a fault implementation class.</p>	<p>Annotation target: TypeProperties: - name Specifies the local name of the XML element that represents the corresponding fault in the WSDL file. You must specify the actual value. (String)- targetNamespace Specifies the namespace of the XML element that represents the corresponding fault in the WSDL file. (String)- faultBean Specifies the name of the fault bean class. (String)</p>	<pre>@Retention(value = RetentionPolicy.RUNTIME) @Target({TYPE}) public @interface WebFault{ String name() default ""; String targetNamespace() default ""; String faultBean() default ""; };</pre>

Parent topic: [JAX-WS annotations reference](#)

JAX-WS Common Annotations (JSR 250)

The JSR 250 specification includes annotations for injecting a resource into an endpoint implementation class, and for managing application lifecycle.

Note: The Java™ class that contains each annotation in the JSR 250 standard is named `javax.annotation.xxx`, where `xxx` is the name of the annotation after the '@' character. For example, the Java class name for the annotation `@Resource` is `javax.annotation.resource`.

Name:	Description:	Properties:	Definitions:
<p><code>@Resource</code></p>	<p>This annotation marks a <code>WebServiceContext</code> resource that the application needs. Apply this annotation to a service endpoint implementation class for a JavaBeans endpoint or a Provider endpoint. The container injects an instance of the <code>WebServiceContext</code> resource into the endpoint implementation when it is initialized.</p>	<p>Annotation target: Field or Method Properties: - authenticationType Indicates the enum that represents the authentication type for this resource. The valid values are APPLICATION or CONTAINER. (String)- description The description of the resource. (String)- mappedName The product-specific name to which this resource is mapped. (String)- name The Java Naming and Directory Interface (JNDI) name of the resource. (String)- shareable The values indicate whether the resource can be shared between this component and other components. The default value is <code>false</code>. (Boolean)- type Indicates the Java type of the resource. (String)</p>	<pre>@Target(value={TYPE, FIELD, METHOD}) @Retention(value=RUNTIME) public @interface Resource { public enum AuthenticationType { APPLICATION, CONTAINER } AuthenticationType authenticationType(); String description(); String mappedName(); String name(); boolean shareable(); Class type(); }</pre>
<p><code>@PostConstruct</code></p>	<p>This annotation marks a method that must be executed after dependency injection is performed on the class. Apply this annotation to a JAX-WS application handler, a service endpoint implementation class for a JavaBeans endpoint or a Provider endpoint.</p>	<p>Annotation target: Method</p>	<pre>@Documented @Retention(value=RUNTIME) @Target(value=METHOD) public @interface PostConstruct { }</pre>

@PreDestroy	<p>This annotation marks a method that must be executed when the instance is in the process of being removed by the container. Apply this annotation to a JAX-WS application handler, a service endpoint implementation class for a JavaBeans endpoint or a Provider endpoint.</p>	<p>Annotation target: Method</p>	<pre>@Documented @Retention(value=RUNTIME) @Target(value=METHOD) public @interface PreDestroy { }</pre>
--------------------	--	--------------------------------------	---

Parent topic: [JAX-WS annotations reference](#)

Rules for methods in classes annotated with @WebService

When you use the @WebService annotation, several rules control how methods are exposed in your web service and how you can use the @WebMethod annotation.

The following rules apply in this situation:

- If the @WebService annotation of an implementation class refers to a Service Endpoint Interface (SEI), the implementation class must have no @WebMethod annotations.
- In an SEI or in an implementation class that does not refer to an SEI, all public methods are treated as exposed, even if the @WebMethod annotation is not specified. The exposed public methods include any inherited methods, except for methods in the Object class.
- In an SEI or in an implementation class that does not refer to an SEI, if a @WebMethod annotation has an attribute of `exclude=true`, the method is not exposed.

Parent topic: [JAX-WS annotations reference](#)

Creating web services and clients with Ant tasks or command line tools

If you prefer not to use the web service wizards, you can use Ant tasks or command line tools to create web services using the IBM® WebSphere® runtime environments or Axis runtime environment.

About this task

The Ant tasks and command line tools support creating web services using both top-down and bottom-up approaches. Once you have created your web service, you can then deploy it to a server, test it, and publish it as a business entity or business service.

- [Creating IBM WebSphere JAX-WS runtime environment web services and clients using Ant tasks](#)

To automate how you create web services and clients for the IBM WebSphere JAX-WS runtime environment, you can use Ant tasks rather than the web services wizard. Run your Ant tasks in the Eclipse workspace or on a command line.

Parent topic: [Developing Web services and clients](#)

Creating IBM WebSphere JAX-WS runtime environment web services and clients using Ant tasks

To automate how you create web services and clients for the IBM WebSphere JAX-WS runtime environment, you can use Ant tasks rather than the web services wizard. Run your Ant tasks in the Eclipse workspace or on a command line.

About this task

Create a WebSphere® JAX-WS web service or client using Ant tasks:

Procedure

1. Create a server to which your web service EAR is deployed: [Creating a JAX-WS-enabled WebSphere server](#).
2. [Create a web project for your web service](#), or create an EJB project, a Java project, or a Java EE application client project for your web service client.
3. [e](#)
4. Customize the Ant properties files for your web service or client:
 - [Ant properties files for top-down Java web services in the IBM WebSphere JAX-WS runtime environment](#)
 - [Ant properties files for bottom-up Java web services in the IBM WebSphere JAX-WS runtime environment](#)
 - [Ant properties files for web service clients for the IBM WebSphere JAX-WS runtime environment](#)
5. Run the Ant task to create the web service or client:
 - [Creating a Java web service for the IBM WebSphere JAX-WS runtime environment using Ant tasks](#)
 - [Creating a web service client for the IBM WebSphere JAX-WS runtime environment using Ant tasks](#)

What to do next

After you create your web service, you can deploy it to a server, test it, and publish it as a business entity or a business service.

- [Customizing the Ant script to run JAX-WS web service Ant tasks in a command line](#)

Before you run the Ant task in a command line to build a web service, you must change the script that specifies how the task runs.

- [Creating a Java web service for the IBM WebSphere JAX-WS runtime environment using Ant tasks](#)

You can use an Ant task instead of the web services wizards to generate a Java web service for the IBM WebSphere JAX-WS runtime environment. Ant tasks support both bottom-up and top-down web services development.

- [Creating a web service client for the IBM WebSphere JAX-WS runtime environment using Ant tasks](#)

If you have a WSDL file, you can use Ant in the Eclipse workspace to generate a web service client with the IBM WebSphere JAX-WS runtime environment.

Parent topic: [Creating web services and clients with Ant tasks or command line tools](#)

Importing Ant files for your JAX-WS web service

To generate a JAX-WS web service using Ant tasks, you must import the Ant task and template properties files into your workspace.

About this task

To import the required Ant files:

Procedure

1. Create a project.
2. In the Enterprise Explorer, right-click your new project and select **File > New > Other > Web Services > Ant Files** and then click **Next**.
3. Select the IBM® WebSphere® JAX-WS runtime environment and the correct web service type for your scenario. Select the project that you created in step 1 as the folder into which the Ant files are imported, and click **Finish**. The XML file that is used to generate the service or client, and the properties file used to customize the service are imported into the `wsgenTemplates` folder.

Scenario of web service	Web service generation file	Properties file
Bottom-up Java™	<code>was_jaxws_bujava.xml</code>	<code>was_jaxws_bujava.properties</code>
Top-down Java	<code>was_jaxws_tdjava.xml</code>	<code>was_jaxws_tdjava.properties</code>
Web service client	<code>was_jaxws_client.xml</code>	<code>was_jaxws_client.properties</code>

What to do next

After you import the required Ant files, you can customize the Ant properties files for your web service or client:

- [Ant properties files for top-down Java web services in the IBM WebSphere JAX-WS runtime environment](#)
- [Ant properties files for bottom-up Java web services in the IBM WebSphere JAX-WS runtime environment](#)
- [Ant properties files for web service clients for the IBM WebSphere JAX-WS runtime environment](#)

Customizing the Ant script to run JAX-WS web service Ant tasks in a command line

Before you run the Ant task in a command line to build a web service, you must change the script that specifies how the task runs.

About this task

To change the script for the Ant task:

Procedure

1. In your product installation directory, go to the `ant` directory under the jar file that contains the `org.eclipse.wst.command.env` plug-in.
 - Windows: this directory might be `C:\<your_WDT_intallation>\plugins\org.eclipse.wst.command.env_X\ant`, where 'X' is the latest version of the plug-in
 - Linux: this directory might be `<your_WDT_installation>/plugins/org.eclipse.wst.command.env_X/ant`, where 'X' is the latest version of the plug-in.
2. Extract the contents of the jar file and edit the file that specifies how the Ant task runs. In Windows, this file is `wsant.bat`; in Linux, this file is `wsant.sh`.
 - A. Set the **JAVAEXE** parameter to the location and name of the JRE `java.exe` file that you want the Ant task to use. For example, in Windows, this value might be `C:\<your_WDT_intallation>\JRE\bin\java.exe`; in Linux, this value might be `<your_java_installation>/jdk/jre/bin/java`.
 - B. Set the **STARTUPJAR** parameter to the location and name of the Eclipse startup JAR file in your product installation. For example, in Windows, this path might be `C:\<your_WDT_intallation>\startup.jar`; in Linux, this path might be `<your_WDT_installation>/startup.jar`.
 - C. Set the **WORKSPACE** parameter to the location and name of the product workspace where you want to build the web service. For example, in Windows, this value might be `C:\workspace`; in Linux, this value might be `/opt/workspace`.
3. Save the script file.
4. Compress the plug-in directory to jar file.

What to do next

After you update the batch file, you can customize the appropriate Ant properties file for the type of web service or client you are creating:

- [Ant properties files for top-down Java web services in the IBM WebSphere JAX-WS runtime environment](#)
- [Ant properties files for bottom-up Java web services in the IBM WebSphere JAX-WS runtime environment](#)
- [Ant properties files for web service clients for the IBM WebSphere JAX-WS runtime environment](#)

Parent topic: [Creating IBM WebSphere JAX-WS runtime environment web services and clients using Ant tasks](#)

Creating a Java web service for the IBM WebSphere JAX-WS runtime environment using Ant tasks

You can use an Ant task instead of the web services wizards to generate a Java™ web service for the IBM® WebSphere® JAX-WS runtime environment. Ant tasks support both bottom-up and top-down web services development.

Before you begin

Before you create a web service, you must complete the following prerequisites:

- Create a server and web project for your web service: [Creating a JAX-WS-enabled WebSphere server](#)
- Import the Ant task and properties files into your workspace: [Importing Ant files for your JAX-WS web service](#)
- Customize the Ant properties files for your web service:
 - [Ant properties files for top-down Java web services in the IBM WebSphere JAX-WS runtime environment](#)
 - [Ant properties files for bottom-up Java web services in the IBM WebSphere JAX-WS runtime environment](#)
- Create or import files into a folder of the web project that you created, according to the type of web service that you want to create:
 - If you want to create a bottom-up service, create or import a bean into the `src` folder
 - If you want to create a top-down service, locate or create a WSDL file, or import one into the project that you created.

About this task

To create the web service, modify the Ant properties file of the service. The name of the Ant properties file depends on the type of web service that you want to create:

- If you want to create a bottom-up service, the name of the Ant properties file is `was_jaxws_bujava.properties`.
- If you want to create a top-down service, the name of the Ant properties file is `was_jaxws_tdjava.properties`.
- If you want to create a client, the name of the Ant properties file is `was_jaxws_client.properties`.

Procedure

Run your Ant task:

- If you want to run the task in the product workspace:
 1. Right-click the imported web services generation XML file and select **Run As > Ant Build**.
 2. In the dialog box, click the **JRE** tab and select **Run in the same JRE as the workspace**.
 3. Click **Apply** and then click **Run**.

After your web service is generated, the console or command line displays a `Build Successful` message.

Results

When your web service is generated, the following files are created depending on the options you selected and the type of web service generated:

- Top-down: The Service Endpoint Interface (SEI). The SEI is the annotated Java representation of the WSDL file for the web service. This interface is used for implementing JavaBeans endpoints or creating dynamic proxy client instances. Data types references in the SEI.
- Bottom-up: The delegate class. This class is a wrapper that contains all the methods from the bean and the JAX-WS annotation the runtime environment recognizes as a web service. If you already have a bean with the `@javax.jws.WebService` annotation before you run the Ant task, this bean is used as-is and new annotations are not added.
- Request/Response/Exception wrapper classes.

Tip: The `wsant.bat` file is installed in the product's `bin` directory. On some systems, this directory might be read-only. In these cases, the Ant task cannot create the output and the web service creation fails. There are two ways to correct this problem:

- Set your `PATH` to point to the `bin` folder, change to a directory where you have write permission, and run the script from there.
- Copy the `wsant.bat` file to a writeable directory.

- Ant properties files for bottom-up Java web services in the IBM WebSphere JAX-WS runtime environment

The `was_jaxws_bujava.properties` file is used to pass data to the Ant tasks when you create a bottom-up Java web service for the IBM WebSphere JAX-WS runtime environment. This file describes the parameter options that are available in the template. Update the template before you run the Ant task to create the web service with the options that you want.

- Ant properties files for top-down Java web services in the IBM WebSphere JAX-WS runtime environment

The `was_jaxws_tdjava.properties` file is used to pass data to the Ant tasks when you create a top-down Java web service for the IBM WebSphere JAX-WS runtime environment. This file describes the parameter options that are available in the template. Update the template before you run the Ant task to create the web service with the options that you want.

Parent topic: [Creating IBM WebSphere JAX-WS runtime environment web services and clients using Ant tasks](#)

Ant properties files for bottom-up Java web services in the IBM WebSphere JAX-WS runtime environment

The `was_jaxws_bujava.properties` file is used to pass data to the Ant tasks when you create a bottom-up Java™ web service for the IBM® WebSphere® JAX-WS runtime environment. This file describes the parameter options that are available in the template. Update the template before you run the Ant task to create the web service with the options that you want.

Required Parameters

-

ScenarioType

- Options: `service`
- This parameter is the type of scenario for the Ant task.

- **InitialSelection**

- This parameter is the URI of the input bean relative to the workspace. This URI has the form `/project/src_folder/package/Class.java`.

Optional Parameters

-

ListRuntimes

- Options: `truefalse`
- If the value is `true`, a list of valid runtime options is displayed when the Ant task is run.

-

ListServers

- Options: `truefalse`
- If the value is `true`, a list of valid server options is displayed when the Ant task is run.

-

Verbose

- Options: `truefalse`
- If the value is `true`, informational messages and errors are displayed when the Ant task is run.

-

Service.RuntimeId

- This parameter is the ID of the runtime environment of the web service. For JAX-WS web services, the only valid value is `com.ibm.ast.ws.jaxws.WasWebServiceRT`.

-

Service.ServerId

- This parameter is the ID of the target server. To set this value automatically, delete or comment out the parameter so that the Ant task receives the correct value from the server settings of the web project. For JAX-WS web services, the only the following values are valid:
 - `com.ibm.ws.ast.st.v7.server.base`
 - `com.ibm.ws.ast.st.v8.server.base`
 - `com.ibm.ws.ast.st.v85.server.base`
 - `com.ibm.ws.st.server.v85.was`

-

ServiceProjectName

- This parameter is a custom name for the web service project. By default, the project takes the name of the bean from which the project is created.

- **ServiceEarProjectName**
 - This parameter is a custom name for the web service EAR project. By default, the project takes the name of the bean from which the project is created.
 - **OverwriteFilesEnabled**
 - Options: `truefalse`
 - This parameter determines whether the Ant task overwrites preexisting files. The default value is `true`.
 - **CreateFoldersEnabled**
 - Options: `truefalse`
 - This parameter determines whether the Ant task creates any required folders. The default value is `true`.
 - **CheckoutFilesEnabled**
 - Options: `truefalse`
 - This parameter determines whether the Ant task checks out files from the repository without prompting. The default value is `true`.
 - **Host**
 - This parameter is the name of the host in the generated WSDL file. The value has the form `myhost:9080`.
 - The default value is `localhost:9080`.
 - **SOAP12**
 - Options: `truefalse`
 - Set this parameter to `true` to enable SOAP 1.2 bindings or `false` to use SOAP 1.1 bindings. The default value is `false`.
 - **MTOM**
 - Options: `truefalse`
 - Set this parameter to `true` to enable SOAP Message Transmission Optimization Mechanism to optimize the transmission of binary content. The default value is `false`.
 - **DelegateClass**
 - This parameter specifies the class name of the Java implementation. The default is the original bean name with "Delegate" appended. The format of this field is in the form of `packageName.ClassNameDelegate`.
 - **JavaToWSDLMapping**
 - Options:
 - `Document Wrapped`
 - `Document Bare`
 - `RPC`
 - The style defines encoding style for messages that are sent to and from the web service. The parameter style determines whether the method's parameters represent the entire message body or whether parameters are elements that are wrapped inside a top-level element that is named after the operation.
 - **GenerateWSDL**
 - Options: `truefalse`
 - Because the annotations in the delegate class are used to tell the runtime environment that the bean is a web service, a static WSDL file is no longer generated into your project automatically. The runtime environment can dynamically generate a WSDL file from the information in the bean. Set this parameter to `true` to generate a WSDL
-

file for the web service. The default value is false.

-

WSDLTargetNamespace

- Enter the WSDL service namespace. This field is only used if GenerateWSDL is set to true.

-

WSDLServiceName

- Enter the WSDL service name. This field is only used if GenerateWSDL is set to true.

-

WSDLPortName

- Enter the WSDL service port. This field is only used if GenerateWSDL is set to true.

-

AddExtension

- Options: truefalse

- Use this option to allow vendor extensions necessary for processing some WSDL documents.

Parent topic: [Creating a Java web service for the IBM WebSphere JAX-WS runtime environment using Ant tasks](#)

Ant properties files for top-down Java web services in the IBM WebSphere JAX-WS runtime environment

The `was_jaxws_tdjava.properties` file is used to pass data to the Ant tasks when you create a top-down Java™ web service for the IBM® WebSphere® JAX-WS runtime environment. This file describes the parameter options that are available in the template. Update the template before you run the Ant task to create the web service with the options that you want.

Required Parameters

- **ScenarioType**

- Options: `service`
- This parameter is the type of scenario for the Ant task.

-

InitialSelection

- This parameter is the URI of the input WSDL file relative to the workspace, or the fully qualified URL of a WSDL file. This URI has the form `/project/MyWSDL.wsdl` or `http://myurl.com/service.wsdl`.

Optional Parameters

- **ListRuntimes**

- Options: `truefalse`
- If the value is `true`, a list of valid runtime options is displayed when the Ant task is run.

- **ListServers**

- Options: `truefalse`
- If the value is `true`, a list of valid server options is displayed when the Ant task is run.

- **Verbose**

- Options: `truefalse`
- If the value is `true`, informational messages and errors are displayed when the Ant task is run.

- **Service.RuntimeId**

- This parameter is the ID of the runtime environment of the web service. For JAX-WS web services, the only valid value is `com.ibm.ast.ws.jaxws.WasWebServiceRT`.

- **Service.ServerId**

- This parameter is the ID of the target server. To set this value automatically, delete or comment out the parameter so that the Ant task receives the correct value from the server settings of the web project. For JAX-WS web services, the only the following values are valid:
 - `com.ibm.ws.ast.st.v7.server.base`
 - `com.ibm.ws.ast.st.v8.server.base`
 - `com.ibm.ws.ast.st.v85.server.base`
 - `com.ibm.ws.st.server.v85.was`

- **JavaOutput**

- Accept the default or enter a path where the Java skeleton is generated.

- **OverwriteFilesEnabled**

- Options: `truefalse`
- This parameter determines whether the Ant task overwrites preexisting files. The default value is `true`.

- **CreateFoldersEnabled**

- Options: `truefalse`
- This parameter determines whether the Ant task creates any required folders. The default value is `true`.

- **CheckoutFilesEnabled**

- Options: truefalse
- This parameter determines whether the Ant task checks out files from the repository without prompting. The default value is true.

- **Host**

- This parameter is the name of the host in the generated WSDL file. The value has the form `myhost:9080`.
- The default value is `localhost:9080`.

-

TargetPackage

- The web services client wizard generates a number of Java files from the specified WSDL. By default it creates a package name that is based on the namespace that is specified in the WSDL file. To override this default behavior, you can specify your own package name.

- **CopyWSDL**

- Options: truefalse
- Set this parameter to true to copy the WSDL file into the service project.

- **EnableWrapperStyle**

- Options: truefalse
- For WSDL documents that implement a document/literal wrapped pattern, a root element is declared in the XML schema and is used as an operation wrapper for a message flow. Separate wrapper element definitions exist for both the request and the response. The default is true.

- **MTOM**

- Options: truefalse
- Set this parameter to true to enable SOAP Message Transmission Optimization Mechanism to optimize the transmission of binary content. The default value is false.

- **GenXSDLLibrary**

- Set this parameter to true to generate a schema library.

- **JAXWSVersion**

- Options: 2.0 2.1
- The Ant task can generate JAX-WS 2.0 or 2.1 compliant code if you are targeting a WebSphere Application Server V7 or V8 server.

- **PortToImplBeanMapping**

- For each port defined in the WSDL file, you can enter a name for the service implementation class in the form of port name to implementation bean value pairs with "," as a delimiter. For example, `Port1=ImplBean1,Port2=ImplBean2`

-

BindingFiles

- If you created JAX-WS or JAXB custom binding files and they are in the workspace, enter their location to use them to create this web service.

- **AddExtension**

- Options: truefalse
- Use this option to allow vendor extensions necessary for processing some WSDL documents.

Parent topic: [Creating a Java web service for the IBM WebSphere JAX-WS runtime environment using Ant tasks](#)

Creating a web service client for the IBM WebSphere JAX-WS runtime environment using Ant tasks

If you have a WSDL file, you can use Ant in the Eclipse workspace to generate a web service client with the IBM® WebSphere® JAX-WS runtime environment.

Before you begin

Before you create the web service client, you must complete the following prerequisites:

- [Create a server for your web service client.](#)
- Optional: You can create a web project, an EJB project, a Java™ project, or a Java EE application client project into which the client code will be generated.
- [Import the Ant task and properties files into your workspace.](#)
- Optional: If you want to run the Ant task from a command line rather than in the workspace, edit the Ant batch file to point to various resources on your system: [Customizing the Ant script to run JAX-WS web service Ant tasks in a command line.](#)
- Customize the Ant properties files for your web service client: [Ant properties files for web service clients for the IBM WebSphere JAX-WS runtime environment.](#)
- Locate or create a WSDL file, or import one into the project that you created.

About this task

In order to create the web service client, you must modify the Ant file and the Ant properties file.

Procedure

1. Edit the `wsgen.xml` Ant file that you imported as a prerequisite for this task. Ensure that the file refers to the client Ant properties file; the Ant file must contain the line `<property file="was_jaxws_client.properties"/>`. Save any changes that you made to the file.
2. Run your Ant task:
 - If you want to run the task in the product workspace:
 - A. Right-click the imported web services generation XML file and select **Run As > Ant Build**.
 - B. In the dialog box, click the **JRE** tab and select **Run in the same JRE as the workspace**.
 - C. Click **Apply** and then click **Run**.

After your web service is generated, the console or command line displays a `Build Successful` message.

Results

When your web service client is generated, the following files are created depending on the options you selected:

- Web service client class and proxy bean.
- Request/Response/Exception wrapper classes.

Tip: The `wsant.bat` file is installed in the product's `bin` directory. On some systems, this directory might be read-only. In these cases, the Ant task cannot create the output and the web service creation fails. There are two ways to correct this problem:

- Set your `PATH` to point to the `bin` folder, change to a directory where you have write permission, and run the script from there.
- Copy the `wsant.bat` file to a writeable directory.

- **Ant properties files for web service clients for the IBM WebSphere JAX-WS runtime environment**

The `was_jaxws_client.properties` file is used to pass data to the Ant tasks when you create a web service client for the IBM WebSphere JAX-WS runtime environment.

Parent topic: [Creating IBM WebSphere JAX-WS runtime environment web services and clients using Ant tasks](#)

Ant properties files for web service clients for the IBM WebSphere JAX-WS runtime environment

The `was_jaxws_client.properties` file is used to pass data to the Ant tasks when you create a web service client for the IBM® WebSphere® JAX-WS runtime environment.

Required Parameters

- **ScenarioType**
 - Options: `client`
 - This parameter is the type of scenario for the Ant task.
- **InitialSelection**
 - This parameter is the URI of the input WSDL file relative to the workspace, or the absolute URL of the WSDL.

Optional Parameters

- **ListRuntimes**
 - Options: `truefalse`
 - If the value is `true`, a list of valid runtime options is displayed when the Ant task is run.
- **ListServers**
 - Options: `truefalse`
 - If the value is `true`, a list of valid server options is displayed when the Ant task is run.
- **Verbose**
 - Options: `truefalse`
 - If the value is `true`, informational messages and errors are displayed when the Ant task is run.
- **Client.RuntimeId**
 - This parameter is the ID of the runtime environment of the web service. For JAX-WS web services, the only valid value is `com.ibm.ast.ws.jaxws.WasWebServiceRT`.
- **Client.ServerId**
 - This parameter is the ID of the target server. To set this value automatically, delete or comment out the parameter so that the Ant task receives the correct value from the server settings of the project.
- **ClientProjectName**
 - This parameter is a custom name for the project for the web service client. By default, the project takes the name of the WSDL file from which the project is created.
- **ClientEarProjectName**
 - This parameter is a custom name for the project for the web service client EAR. By default, the project takes the name of the WSDL file from which the project is created. (Set this parameter only if it applies to the server type.)
- **ClientComponentType**
 - Options: `template.jst.webtemplate.jst.ejb, template.jst.appclienttemplate.jst.utility`
 - This parameter is the type of the web service client. The default value is `template.jst.web`.
- **CreateFoldersEnabled**
 - Options: `truefalse`
 - This parameter determines whether the Ant task creates any required folders. The default value is `true`.
- **CheckoutFilesEnabled**
 - Options: `truefalse`
 - This parameter determines whether the Ant task checks out files from the repository without prompting. The default value is `true`.
- **TargetPackage**
 - The web services client wizard generates a number of Java™ files from the specified WSDL. By default it creates a package name that is based on the namespace that is specified in the WSDL file. To override this default behavior,

you can specify your own package name.

- **GenPortableClient**

- Set this parameter to `True` to enable you move your web service client code from one machine to another or from one instance of WebSphere Application Server to another. If this option is selected, the WSDL document and all the XML schema and other WSDL documents that it depends upon are copied into the client project under `WEB-INF/wsdl`. A `file:relativeURL` pointing to this copy is then injected into the JAX-WS Service class's static initialization block.

- **JAXWSVersion**

- Options: 2.0 2.1

- The Ant task can generate JAX-WS 2.0 or 2.1 compliant code if you are targeting a WebSphere Application Server V7 server.

- **AsyncOperation**

- Options: `truefalse`

- Set this parameter to `true` to enable an asynchronous client. For each method in the web service two more methods are created. These methods are polling and callback methods that allow the client to function asynchronously. The default is `false`.

- **PortToProxyMapping**

- You can accept the default proxy name or specify a proxy class name for each port. The syntax is in the form of a port name to proxy class name-value pair, delimited by `,`. For example, `Port1=ProxyClass1,Port2=ProxyClass2`

- **BindingFiles**

- If you created JAX-WS or JAXB custom binding files and they are in the workspace, enter their location to use them to create this web service.

- **AddExtension**

- Options: `truefalse`

- Use this option to allow vendor extensions necessary for processing some WSDL documents.

Parent topic:[Creating a web service client for the IBM WebSphere JAX-WS runtime environment using Ant tasks](#)

Web services: Editing, assembling, and securing tasks

After you create a web service or client, you can do various assembly tasks, such as editing the web service deployment descriptors, adding handlers, and enabling security.

Parent topic: [Developing web service applications](#)

Creating and editing JAX-WS web service handlers

You can add JAX-WS logical or protocol handlers to intercept inbound and outbound messages to or from web services and their clients. You can select from any currently available JAX-WS web services and clients and start the Handler Creation Wizard. In the wizard, you provide the class name of the handler, the handler name, and an optional display name, and specify the type of handler. When finished, the wizard generates the skeleton handler code and updates the applicable deployment descriptor.

About this task

The Handler Creation Wizard helps you:

- Add or edit service-side handlers
- Add or edit client-side handlers

Parent topic: [Developing web service applications](#)

Adding or editing service-side handlers

Procedure

1. Open the Services view.
2. From the JAX-WS tree, select your web service. Right-click and select **Configure JAX-WS Web Service Handlers**.
3. If handlers are already defined for the service, you can delete or reorder them. You can also add a handler by clicking **Add**.
 - A. Select if you want to create a new handler class or select an existing one.
 - B. Enter or browse to the fully qualified class name for the handler.
 - C. Enter a name for the handler.
 - D. Optional: Enter a display name for the handler.
 - E. Select the handler type, logical or protocol.
 - F. The output folder for the handler is listed. The default folder is the `src` folder for the service.

Adding client-side handlers

About this task

Procedure

1. Open the Services view.
2. From the JAX-WS tree, select your web service client. Right-click and select **Configure JAX-WS ClientHandlers**.
3. If handlers are already defined for the client, you can delete or reorder them. You can also add a handler by clicking **Add**.
 - A. Browse to the fully qualified class name for the handler.
 - B. Enter a name for the handler.
 - C. Optional: Enter a display name for the handler.
 - D. Select the handler type, logical or protocol.

E. The output folder for the handler is listed. The default folder is the `src` folder for the client.

Testing web service handlers Procedure

1. Modify the generated handler classes (such as by adding `println`).
2. Run the web service client sample JSP and test the web services to verify that the handlers intercept the traffic correctly.

Merging web services updates

After you create a web service, you might want to change it. Although you cannot automatically propagate all your changes to all the required files, to retain your changes while you update the web service, you can merge a generated skeleton file. You can then regenerate your web service, and your changes remain intact.

The merge generated skeleton file option can be enabled in the web service preferences on the Resource Management page. This option is enabled by default. If it is enabled, when a skeleton file of the same name exists in the workspace, the newly generated skeleton file merges with the existing skeleton file. Enable this option to preserve code that you modified in the existing skeleton files.

Note: When this preference is enabled, if you change the return type of an interface, the resulting skeleton file has a method that returns the new interface type but has a method body from the original skeleton file, which returns a different type. This change results in a compilation error that you must correct manually.

Note: This function does not merge annotations in JAX-WS web services. The annotations in the new skeleton file are discarded, and the existing annotations remain untouched.

Parent topic: [Developing web service applications](#)

Securing web services

Web services security for WebSphere® Application Server is based on the OASIS web services security (WSS) Version 1.0 specification, the Username token Version 1.0 profile, and the X.509 token Version 1.0 profile. These standards and profiles address how to provide protection for messages that are exchanged in a web service environment.

- [Web services security overview](#)

The OASIS web services security (WSS) Version 1.0 specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message. WSS is a message-level standard that is based on securing SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens. New in the Feature Pack for Web Services, JAX-WS web services can be easily secured using policy sets.

- [Qualities of service for JAX-WS web services and clients](#)

You can use policy sets to simplify configuring the qualities of service for web services and clients. Policy sets are assertions about how web services are defined. Using policy sets, you can combine configurations for different policies. You can use policy sets with JAX-WS applications, but not with JAX-RPC applications.

- [Creating a policy set attachment on the client side](#)

You can add security to a web service client by attaching policy sets to the client. Each attachment specifies an endpoint, a policy set, and a binding. Because each configuration is specific to an application and a user, you must configure a binding for some policy types.

- [Creating a policy set attachment on the server side](#)

You can add security to a web service by attaching policy sets to the service. Each attachment specifies an endpoint, a policy set, and a binding.

- [Modifying policy set attachments](#)

After you create policy set attachments for your web service or client, you can modify attachment attributes. For example, you can specify different endpoints and policy sets. For policy set attachments on the client side, you can also specify different bindings and binding configurations.

- [Editing policy set bindings](#)

You can create and edit existing policy set binding configurations by using the binding stand-alone editors.

- [Importing policy sets into your workspace](#)

In addition to using the policy sets that come with the product, you can import policy sets that were exported from a server. After you import these policy sets into your workspace, you can attach them to your web services and clients.

Parent topic: [Developing web service applications](#)

Web services security overview

The OASIS web services security (WSS) Version 1.0 specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message. WSS is a message-level standard that is based on securing SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens. New in the Feature Pack for Web Services, JAX-WS web services can be easily secured using policy sets.

Important: Applicable edition: Full profile

To secure web services, you must consider a broad set of security requirements, including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, federation, delegation, and auditing across a spectrum of application and business topologies. One of the key requirements for the security model in today's business environment is the ability to interoperate between formerly incompatible security technologies, such as public key infrastructure and Kerberos, in heterogeneous environments such as Microsoft .NET and Java™ (Java EE). The complete web services security protocol stack and technology roadmap is described in [Security in a Web Services World: A Proposed Architecture and Roadmap](#) .

The [Web Services Security: SOAP Message Security 1.0 specification](#) outlines a standard set of SOAP extensions that you can use to build secure web services. These standards confirm integrity and confidentiality, which are provided with digital signature and encryption technologies. In addition, web services security provides a general-purpose mechanism for associating security tokens with messages. A typical example of the security token is a Username token, in which a user name and password are included as text. web services security defines how to encode binary security tokens using methods such as X.509 certificates and Kerberos tickets. However, the required security tokens are not defined in the web service security Version 1.0 specification. Instead, the tokens are defined in separate profiles such as the Username token profile, the X.509 token profile, the SAML profile, the Kerberos profile, and the XrML profile.

Enabling JAX-WS web services security using policy sets

In the V6.1 Feature Pack for Web Services and V7, WebSphere® Application Server uses the policy set model for implementing the WSS Version 1.1 specification, the Username token Version 1.1 profile, and the X.509 token Version 1.1 profile. Policy sets combine configuration settings, including settings for transport and message level configuration, such as WS-Addressing, WS-ReliableMessaging, WS-SecureConversation, and WS-Security.

To enable policy sets for your web services, refer to [Managing policy sets for JAX-WS web services and clients](#).

Additionally, the following information demonstrates policy sets.

- [Tutorial: Creating a secured JAX-WS web service from an WSDL file](#)

The following book contains a chapter on policy sets in the WebSphere Application Server V6.1 Feature Pack for Web Services: [IBM® Redbooks®: Web Services Feature Pack for WebSphere Application Server V6.1](#)

More information on securing web services

In addition to the information provided in this documentation, several useful tutorials and articles that demonstrate how to secure web services using WebSphere Application Server are available on developerWorks®.

- [Message-level security with JAX-WS on WebSphere Application Server V7: Using Rational® Application Developer V7.5.2 to build secure JAX-WS web services](#)

Parent topic: [Securing web services](#)

Related concepts:

[Qualities of service for JAX-WS web services and clients](#)

Qualities of service for JAX-WS web services and clients

You can use policy sets to simplify configuring the qualities of service for web services and clients. Policy sets are assertions about how web services are defined. Using policy sets, you can combine configurations for different policies. You can use policy sets with JAX-WS applications, but not with JAX-RPC applications.

Important: Applicable edition: Full profile

For information about web services security for WebSphere® Application Server, see: [Administering web services - Security \(WS-Security\)](#).

For information about web policy sets for WebSphere Application Server, see: [Administering web services - Policy \(WS-Policy\)](#).

The following developerWorks® tutorial provides a detailed example of how to configure message-level security for the SOAP message by configuring policy sets in the workbench: [Message-level security with JAX-WS on WebSphere Application Server V7: Using Rational® Application Developer V7.5.2 to build secure JAX-WS web services](#)

For information about configuring web services, see: chapter 14 in the [Rational Application Developer for WebSphere Software V8 Programming Guide](#)

A policy set is identified by a unique name. An instance of a policy set consists of a collection of policy types. An empty policy set has no defined policy instance.

You can use the policy sets that are included with this product to simplify configuring the qualities of service for your web services and clients. For example, the Reliable Secure Profile (RSP) default policy set consists of instances of the WS-SecureConversation and WS-Securitypolicy types. For more information about the policy sets that are included with this product, see the related concepts.

Policies are defined based on a quality of service. Policy definitions are typically based on WS-Policy standards language. For example, the WS-Security policy is based on the current WS-SecurityPolicy language from the Organization for the Advancement of Structured Information Standards (OASIS) standards.

Policy sets omit application or user-specific information, such as keys for signing, keystore information, or persistent store information. Instead, application and user-specific information is defined in the bindings. Typically, bindings are specific to the application or the user, and bindings are not normally shared. On the server side, if you do not specify a binding for a policy set, a default binding is used for that policy set. On the client side, you must specify a binding for each policy set. A policy set attachment defines which policy set is attached to service resources, and which bindings are used for the attachment. The bindings define how the policy set is attached to the resources. An attachment is defined outside of the policy set, as metadata associated with the application. To enable a policy set to work with an application, a binding is required. Use the policy set attachment wizards to configure bindings.

Policy sets can be created, deleted, copied, imported, or exported within WebSphere Application Server using either the administrative console or the wsadmin commands. You can then import or export policy sets from the workbench using the **Import > Web services > WebSphere Policy Sets** or **Export > Web services > WebSphere Policy Sets** wizards. Policy sets are then attached to web services and clients using the **Manage Policy Set Attachments** wizards.

Parent topic: [Securing web services](#)

Related concepts:

[Web services security overview](#)

Web services policies

Using web services policies, you can define the qualities of service for your web services and clients.

Important: Applicable edition: Full profile

The product provides several types of web services policy:

- **WS-Addressing**

- Based on the World Wide Web Consortium (W3C) WS-Addressing specifications for web services. This family of specifications provides transport-neutral mechanisms to address web services and to facilitate end-to-end addressing. This specification provides asynchronous support.

- **WS-Security**

- Based on the WS-Secure Conversation (WS-SC) and WS-Security specifications, as well as the associated token profiles. The WS-Security specification and its associated token profiles define a way to send security tokens and provide message integrity and confidentiality. The WS-Secure Conversation specification establishes a secure context, based on shared keys, for the client and server to use for a series of messages. This standard provides a framework across organizations that defines how to secure the entire conversation. Use the WS-Security policy to define how the SOAP messages are secured. You can also use WS-Security policies to define the bootstrap policy that is used to acquire security context tokens. Security context tokens are used by secure conversation.

- **WS-Reliable Messaging (WS-RM)**

- Enables the sender and receiver to assure the quality of services in a set of messages. With this policy, you can address latency issues, maintenance interruption, and other problems that prevent messages from being completed. This quality assurance is critical for stateful applications.

- **WS-Transaction**

- Provides support for atomic transactions in a web services environment. This quality of service enables web service applications and the resources that they use to participate in distributed global transactions.

- **HTTP Transport**

- Applies HTTP features and HTTP connections policies to outbound messages. The response listener policy is enforced on inbound messages.

- **SSL Transport**

- Provides SSL transport security for the HTTP protocol with web services applications.

Web services policy sets that are included with the product

You can use the policy sets in this product to simplify configuring the qualities of service for your web services. Using these policy sets, you can combine configurations for different policies.

Important: Applicable edition: Full profile

WebSphere Application Server V7.0.0.7 and later policy sets

Starting in WebSphere® Application Server V7 Fix Pack 7, you can secure web services with Security Assertion Markup Language (SAML). Use SAML assertions to represent user identity and user security attributes, and optionally, to sign and to encrypt SOAP message elements. WebSphere Application Server supports SAML assertions using the bearer subject confirmation method and the holder-of-key subject confirmation method as defined in the OASIS WSS SAML Token Profile Version 1.1 specification. Policy sets and general bindings that support SAML are included with the product SAML function. To use SAML assertions, you must modify the provided sample general binding.

Tip: Read the WebSphere Application Server documentation on SAML policy sets before you apply them to your web service. This documentation describes how SAML is supported within WebSphere Application Server and what limitations exist. See [Securing Web services using Security Assertion Markup Language \(SAML\)](#).

WebSphere Application Server V7.0 and V8.0 policy sets

The following WebSphere Application Server V7.0 and V8.0 policy sets are included with the product:

- Kerberos V5 HTTPS default

- This policy set provides message authentication with a Kerberos Version 5 token. Message integrity and confidentiality are provided by Secure Sockets Layer (SSL) transport security. This policy set follows the OASIS Kerberos Token Profile V1.1 and WS-Security specifications. When you use this policy set, configure the basic authentication data and custom properties such as the `com.ibm.wsspi.wssecurity.krbtoken.targetServiceName` and `com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost` custom properties in the client bindings. For more information, see the Authentication generator or consumer token settings and Protection token settings (generator or consumer) topics.

- LTPA WSSecurity default

- This policy set provides the following features:
 - Message integrity through digital signature (using RSA public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using WS-Security specifications.
 - Message confidentiality through encryption (using RSA public-key cryptography) to encrypt the body, signature, and signature elements using WS-Security specifications.
 - A Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service.

- SSL WSTransaction

- Use this policy set to coordinate distributed transactional work atomically, interoperably and securely, by using the WS-AtomicTransaction specification and SSL Transport security. Also, use this policy set to coordinate loosely coupled business processes, with the ability to compensate actions if a failure occurs in the business activity using the WS-BusinessActivity specification and SSL Transport security.

- Username SecureConversation

- This policy set provides the following features:
 - Message integrity through digital signature that includes signing the body, time stamp, and WS-Addressing headers using WS-SecureConversation and WS-Security specifications
 - Message confidentiality through encryption that includes encrypting the body, signature and signature confirmation elements, using WS-SecureConversation and WS-Security specifications

- A user name token included in the request message to authenticate the client to the service. The user name token is encrypted in the request

- Username WSSecurity default

- This policy set provides the following features:
 - Message integrity by digital signature (using RSA public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using WS-Security specifications
 - Message confidentiality by encryption (using RSA public-key cryptography) to encrypt the body, signature, and signature confirmation elements using WS-Security specifications
- A user name token included in the request message to authenticate the client to the service. The user name token is encrypted in the request

- WS-I RSP

- This policy set enables unmanaged non-persistent WS-ReliableMessaging, which provides the ability to deliver a message reliably to its intended receiver. This policy set works only in a single-server environment and does not work in a clustered environment. Message integrity is provided by digitally signing the body, the time stamp, and the WS-Addressing headers. Message confidentiality is provided by encrypting the body and the signature. This policy set follows the WS-SecureConversation and WS-Security specifications. Web Services Reliable Messaging policy sets must be applied at the service level for the Reliable Messaging quality of service to be respected by the runtime environment. Policy sets that are applied at the endpoint level or the operation level are ignored by the runtime environment.

- WSAddressing default

- The WSAddressing default policy set provides a transport-neutral way to uniformly address web services and messages. The WSAddressing default policy set is based on the WS-Addressing specification. The WS-Addressing standard uses endpoint references and message-addressing properties to facilitate the addressing of web services in a standard and interoperable way. Use the WSAddressing default policy set as provided with the application server. To customize the policy set, you must first copy the policy set, and then configure custom policy settings and bindings to meet your needs.

- WSHTTPS default

- This policy set provides SSL transport security for the HTTP protocol with web services applications.

- WSReliableMessaging persistent

- This policy set enables both WS-ReliableMessaging and WS-Addressing and uses the maximum quality of service, managed persistent. This quality of service supports asynchronous web service invocations and uses a service integration messaging engine and message store to manage the sequence state. Messages are processed within transactions and persisted at the web service requester server and at the web service provider server, and they are recoverable if the server fails. In-order delivery is set to "false", so messages are not necessarily delivered in the order in which they were sent. Because this policy set specifies managed persistent quality of service, you must define bindings to the service Integration Bus and messaging engine that you want to use to manage the WS-ReliableMessaging state. You can attach and bind a WS-ReliableMessaging policy set to a web service application by using the administrative console or the wsadmin tool.

Web Services Reliable Messaging policy sets must be applied at the service level for the Reliable Messaging quality of service to be respected by the runtime environment. Policy sets that are applied at the endpoint level or the operation level are ignored by the runtime environment.

WebSphere Application Server V6.1 policy sets

The following WebSphere Application Server V6.1 policy sets are included with the product:

- RAMP default policy sets

- **RAMP default**

- Default Reliable Asynchronous Messaging Profile (RAMP) 1.0. This policy set provides the following features:
 - Reliable message delivery to the intended receiver by enabling WS-ReliableMessaging
 - Message integrity by digital signature that includes signing the body, time stamp, WS-Addressing headers, and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications
 - Confidentiality by encryption that includes encrypting the body, signature and signature confirmation elements, using the WS-SecureConversation and WS-Security specifications

- LTPA RAMP default

- This policy set provides the following features:
 - Reliable message delivery to the intended receiver by enabling WS-ReliableMessaging
 - Message integrity by digital signature that includes signing the body, time stamp, WS-Addressing headers, and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications
 - Confidentiality by encryption that includes encrypting the body, signature and signature confirmation elements, using the WS-SecureConversation and WS-Security specifications
 - A Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service

- Username RAMP default

- This policy set provides the following features:
 - Reliable message delivery to the intended receiver by enabling WS-ReliableMessaging
 - Message integrity by digital signature that includes signing the body, time stamp, WS-Addressing headers, and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications
 - Confidentiality by encryption that includes encrypting the body, signature and signature confirmation elements, using the WS-SecureConversation and WS-Security specifications
 - A user name token included in the request message to authenticate the client to the service. The user name token is encrypted in the request

- SecureConversation policy sets

- SecureConversation

- This policy set provides the following features:
 - Message integrity by digital signature that includes signing the body, time stamp, and WS-Addressing headers using WS-SecureConversation and WS-Security specifications
 - Message confidentiality by encryption that includes encrypting the body, signature and signature confirmation elements, using WS-SecureConversation and WS-Security specifications

- LTPA SecureConversation

- This policy set provides the following features:
 - Message integrity by digital signature that includes signing the body, time stamp, and WS-Addressing headers using WS-SecureConversation and WS-Security specifications
 - Message confidentiality by encryption that includes encrypting the body, signature and signature confirmation elements, using WS-SecureConversation and WS-Security specifications
 - A Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service

- Username SecureConversation

- This policy set provides the following features:
 - Message integrity by digital signature that includes signing the body, time stamp, and WS-Addressing headers using WS-SecureConversation and WS-Security specifications
 - Message confidentiality by encryption that includes encrypting the body, signature and signature confirmation elements, using WS-SecureConversation and WS-Security specifications
 - A user name token included in the request message to authenticate the client to the service. The user name token is encrypted in the request

- WSReliableMessaging policy sets

- WSReliableMessaging default

- This policy set enables both WS-ReliableMessaging and WS-Addressing, and the policy set uses the minimum quality of service unmanaged non-persistent. This quality of service requires minimal configuration. However, it is non-transactional and, although it allows for the resending of messages that are lost in the network, failure of a server results in lost messages. This quality of service is for single server only; it does not work in a cluster.

- WSReliableMessaging 1_0

- This policy set enables both WS-ReliableMessaging Version 1.0 and WS-Addressing, and it uses the minimum quality of service unmanaged non-persistent. This quality of service requires minimal configuration. This quality of service is non-transactional, however. Although it allows for the resending of messages that are lost in the network, failure of a server results in lost messages. This quality of service is for single-server use only; it does not work in a cluster. You can use this policy set with .NET-based web services.

- WSReliableMessaging persistent

- This policy set enables both WS-ReliableMessaging and WS-Addressing, and the policy set uses the maximum quality of service managed persistent. This quality of service supports asynchronous web service invocations, and uses a service integration messaging engine and message store to manage the sequence state. Messages are processed within transactions and are persisted at the web service requester server and at the web service provider server. The messages are recoverable if the server fails.

Web Services Reliable Messaging policy sets must be applied at the service level for the Reliable Messaging quality of service to be respected by the runtime environment. Policy sets that are applied at the endpoint level or the operation level are ignored by the runtime environment.

- WSSecurity default policy sets

- WSSecurity default

- This policy set provides the following features:
 - Message integrity by digital signature (using RSA public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using WS-Security specifications
 - Message confidentiality by encryption (using RSA public-key cryptography) to encrypt the body, signature and signature confirmation elements using WS-Security specifications

- LTPA WSSecurity default

- This policy set provides the following features:
 - Message integrity by digital signature (using RSA public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using WS-Security specifications
 - Message confidentiality by encryption (using RSA public-key cryptography) to encrypt the body, signature, and signature confirmation elements using WS-Security specifications
 - A Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service

- Username WSSecurity default

- This policy set provides the following features:
 - Message integrity by digital signature (using RSA public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using WS-Security specifications
 - Message confidentiality by encryption (using RSA public-key cryptography) to encrypt the body, signature, and signature confirmation elements using WS-Security specifications
 - A user name token included in the request message to authenticate the client to the service. The user name token is encrypted in the request

- WSTransaction policy sets

- WSTransaction

- This policy set enables WS-Transaction, which provides the ability to coordinate distributed transactional work atomically and interoperably using the WS-AtomicTransaction specification.

- SSL WSTransaction

- This policy set enables WS-Transaction, which provides the ability to coordinate distributed transactional work atomically, interoperably and securely using the WS-AtomicTransaction specification and SSL Transport security.

- Other default policy sets

- WSAddressing default

- This policy set enables WS-Addressing support, which uses endpoint references and message-addressing properties to facilitate the addressing of web services in a standard and interoperable way.

- WSHTTPS default

- This policy set provides SSL transport security for the HTTP protocol with web services applications.

WebSphere Application Server V7.0 and V8.0 system policy sets

The following WebSphere Application Server V7.0 and V8.0 system policy sets are included with the product:

- SystemWSSecurityDefault

- This system policy set specifies the asymmetric algorithm and both the public and private keys to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using RSA encryption. Message confidentiality is provided by encrypting the body and signature using RSA encryption. This policy set follows the WS-Security specifications for the issue and renew trust operation requests.

Creating a policy set attachment on the client side

You can add security to a web service client by attaching policy sets to the client. Each attachment specifies an endpoint, a policy set, and a binding. Because each configuration is specific to an application and a user, you must configure a binding for some policy types.

Before you begin

Important: Applicable edition: Full profile

You created a web service client that is in your workspace.

About this task

For a web service and a client of that service, the policy sets and bindings configuration must match for the service to function correctly. Use the information that you obtained from your web service provider when you specify policy sets and most bindings configuration on the client side.

To create a policy set attachment for your web service client:

Procedure

1. In the Services view, select your web service client object, right-click, and select **Manage Policy Set Attachment**.
2. For each attachment that you want to create, click **Add** and use the Configure Policy Set and Binding window to [attach a policy set to an endpoint and specify a binding](#). Each attachment that you create is listed in the **Application** table.
3. To configure the bindings for an attachment:
 - A. Select the attachment in the **Application** table.
 - B. In the **Binding Configurations** table, for each binding that has a status of `Binding not configured`, select the binding, click **Configure**, and use the binding configuration window to [configure a binding](#). The window that opens depends on the type of binding that you select.
4. After you configure all of the bindings that require configuration, each binding in the **Binding Configurations** table has a status of `Binding Configured` or `N/A` (not applicable). Click **Finish** to complete the wizard.

Results

A `clientPolicyAttachments.xml` is created in the `META-INF` folder of your web service application (EAR).

For more information about creating a policy set attachment, see the related tasks.

- [Attaching a policy set to an endpoint](#)

When you create or modify a policy set attachment for a web service or client, you can associate a policy set with an endpoint. For a client, you can also specify a binding.

- [Configuring bindings for policy types](#)

After you attach a policy set to a web services endpoint, you must configure bindings for several policy types to customize the types for a particular environment and platform.

Parent topic: [Securing web services](#)

Attaching a policy set to an endpoint

When you create or modify a policy set attachment for a web service or client, you can associate a policy set with an endpoint. For a client, you can also specify a binding.

Before you begin

Important: Applicable edition: Full profile

You must be in the Configure Policy Set and Binding window of a policy set attachment wizard.

About this task

You can use this window more than once to associate different policy sets with different levels of a web service. A web service has three levels of generality: service, endpoint, and operation. The service level is the most general, and the operation level is the most specific. The most specific attachment that applies is used for a certain invocation of a web service. For example, if you create attachments to both a web service and an operation in that service, invocations of the operation use the attachment for the operation, but invocations of other operations use the attachment for the service.

Procedure

1. From the **Service Name** list, select the web service that you want to secure.
2. Optional: If you want finer-grained control, from the **Endpoint** list, select the name of the endpoint that you want to secure. An endpoint is also known as a port type.
3. Optional: If you want even finer-grained control, from the **Operation** list, select the name of the operation that you want to secure.
4. From the **Policy Set** list, select the policy set that you want to attach to the web service, endpoint, or operation that you selected in the **Application** section. The policy set that you attach to a web service on the client side must match the policy set that is attached to the web service on the server side. For the client, use the information from the web service provider. If you set a global or project level preference for a default policy set, this policy set is pre-selected for you.
5. To specify a binding for a client-side attachment, in the **Binding** list, select or enter a name for the binding that you want to associate with your attachment. Because the information that a binding contains is unique to a specific environment or platform, you can use each binding with only one attachment. You do not need to specify a binding for a server-side attachment.
6. Click **OK**.

Results

Your attachment is shown in the **Application** table of the policy set attachment wizard.

Parent topic: [Creating a policy set attachment on the client side](#)

Configuring bindings for policy types

After you attach a policy set to a web services endpoint, you must configure bindings for several policy types to customize the types for a particular environment and platform.

Before you begin

Important: Applicable edition: Full profile

Read about [web service bindings](#).

About this task

The bindings configuration of a web service on the client side must match the bindings configuration of the web service on the server side. For the client, use information from the web service provider to match the values that are specified on the server side.

Procedure

1. Open a binding configuration window of a policy set attachment wizard.
2. Configure bindings for the policy types that you want the web service client to use.

- [Configuring bindings for the HTTP transport policy](#)

To use the HTTP transport policy with your web service clients, you must first configure bindings for the policy.

- [Configuring bindings for the SSL transport policy](#)

To use the SSL transport policy with your web service clients, you must first configure bindings for the policy.

- [Configuring a binding for the WS-Reliable Messaging policy](#)

To use the WS-Reliable Messaging policy with your web service clients, you must first configure bindings for the policy.

- [Configuring a binding for the WS-Security policy](#)

To use the WS-Security policy with your web service clients, you must first configure bindings for the policy.

Parent topic: [Creating a policy set attachment on the client side](#)

Web services bindings

A set of bindings is a named object that is associated with a specific policy set and service resource that is attached to the policy set. Bindings contain information that is specific to an environment and platform.

Important: Applicable edition: Full profile

Bindings contain the following types of information:

- Keys that are used for signature and encryption
- Keystore information
- Authentication information
- Persistent information

Typically, bindings are specific to an application or a platform, which implies that bindings are not normally shared. There exists a set of default bindings that can be used by all policy sets. Custom bindings, however, are defined within an application. One binding is associated with one reference to a policy set; this association is a one-to-one relationship.

You must configure a binding for each of these policy types:

- HTTP Transport
- SSL Transport
- WS-Reliable Messaging
- WS-Security

You do not need to configure a binding for the following policy types:

- WS-Addressing
- WS-Transaction

These two policy types have no properties that can be configured. In a policy set, these policy types are either included and enabled, or excluded or disabled.

Configuring bindings for the HTTP transport policy

To use the HTTP transport policy with your web service clients, you must first configure bindings for the policy.

About this task

Important: Applicable edition: Full profile

With HTTP, your web services and clients can communicate via messages in the Simple Object Access Protocol (SOAP).

SOAP is used by web services that support two specifications, Web Services for Java™Platform, Enterprise Edition (J2EE) and Java API for XML-based Remote Procedure Call (JAX-RPC). Most web services use HTTP transport.

To configure a binding for the HTTP transport policy:

Procedure

1. In the Client Side Policy Set Attachment wizard, select the HTTPTransport policy type in the **Bindings Configuration** table; then click **Configure**.
2. Under **Proxy for outbound service requests**, specify the host name and port number of the proxy for your web service client and the user name and password for the proxy.
3. Under **Basic authentication for outbound service requests**, specify the user name and password for basic authentication.
4. Click **OK**.

Parent topic: [Configuring bindings for policy types](#)

Configuring bindings for the SSL transport policy

To use the SSL transport policy with your web service clients, you must first configure bindings for the policy.

About this task

Important: Applicable edition: Full profile

Secure Sockets Layer (SSL) transport adds security to the HTTP protocol with your web services applications.

Procedure

1. In the Client Side Policy Set Attachment wizard, select the SSLTransport policy type in the **Bindings Configuration** table; then click **Configure**.
2. Under **Outbound service requests**, specify the settings and property file that are used for outbound service requests.
3. Under **Inbound service responses**, specify the settings and property file that are used for inbound service responses.
4. Click **OK**.

Parent topic: [Configuring bindings for policy types](#)

Configuring a binding for the WS-Reliable Messaging policy

To use the WS-Reliable Messaging policy with your web service clients, you must first configure bindings for the policy.

About this task

Important: Applicable edition: Full profile

The WS-ReliableMessaging policy is an interoperability standard for the reliable transmission of messages between two endpoints of a web service. For your web services that use SOAP over HTTP, you can use WS-ReliableMessaging to make these services interoperable and reliable, without needing to write custom code. For this policy, managed qualities of service are supported by a service integration bus.

To configure a binding for the WS-Reliable Messaging policy:

Procedure

1. In the Client Side Policy Set Attachment wizard, select the WSReliableMessaging policy type in the **Bindings Configuration** table; then click **Configure**.
2. In the **Bus Name** field, type the name of the service integration bus that is used for managed qualities of service.
3. In the **Messaging Engine Name** field, type the name of the messaging engine (or bus member) that is used for managed qualities of service.
4. Click **OK**.

Parent topic: [Configuring bindings for policy types](#)

Configuring a binding for the WS-Security policy

To use the WS-Security policy with your web service clients, you must first configure bindings for the policy.

About this task

Important: Applicable edition: Full profile

The WS-Security specification includes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. This specification provides protection for a message by encrypting or digitally signing (or both) a message body, headers, attachment, or any combination (or parts) of these elements. The specification also provides a mechanism for associating security tokens with messages.

To configure a binding for the WS-Security policy:

Procedure

1. In the Client Side Policy Set Attachment wizard, select the WSSecurity policy type in the **Bindings Configuration** table; then click **Configure**.
2. On the Digital Signature (bootstrap) tab:
 - A. Under **Outbound Message Security Configuration**, select the type of information that your key contains and the algorithm that is used to transform your outbound messages that have digital signatures. Use **Callback Handler Settings** to specify settings for your keystore.
 - B. Under **Inbound Message Security Configuration**, select the algorithm that is used to transform your outbound messages that have digital signatures. Select **Callback Handler Settings** to specify settings for your keystore. Within this window, select **Trust Any Certificate** if you want to accept all incoming messages that have digital signatures, without verifying credentials. If you clear this check box, you can specify your keystore settings and optionally specify a certificate in the **Certificate Path** field.
3. On the XML Encryption (bootstrap) tab:
 - A. Under **Outbound Message Security Configuration**, select the type of information that your key contains and the algorithm that is used to transform your outbound messages that have digital signatures. Use **Callback Handler Settings** to specify settings for your keystore. Select **Enable MTOM WS-Security Optimization** if you want to use the SOAP Message Transmission Optimized Mechanism (MTOM) when you send binary data with your messages. Select **Enable Encrypted Header for WS-Security 1.0** if you want to use encrypted SOAP headers in the WS-Security version 1.0 specification format.
 - B. Under **Inbound Message Security Configuration**, use **Callback Handler Settings** to specify settings for your keystore.
4. Select **Enable Message Expiration** if you want to enable expiration of your sent messages. Type the number of minutes after which your sent messages expire in the **Message Expiration Interval** field. This number must be a positive integer. By default, sent messages remain permanently valid.
5. Click **OK**.

What to do next

Note: The window displays read-only information about the token types, callback handlers, and JAAS logins in the binding to help you with specifying the required values.

Parent topic: [Configuring bindings for policy types](#)

Specifying keystore settings for bindings configuration

The WS-Security policy uses keys to digitally sign messages or encrypting your web service messages. As part of configuring a binding for the WS-Security policy, you must provide information about the files in which these keys are stored.

Before you begin

Important: Applicable edition: Full profile

Your file system must contain keystores that can be used to support the WS-Security policy. Obtain these files from your web service provider.

Procedure

1. In the Client Side Policy Set Attachment wizard, select the WS-Security policy type in the **Bindings Configuration** table; then click **Configure**.
2. In the WS-Security binding configuration window, the Digital Signature (bootstrap) tab and the XML Encryption (bootstrap) tab both contain areas where you can specify settings for outbound and inbound messages. Click **Callback Handler Settings** in the area where you want to specify keystore settings.
3. Enter the callback handler settings in the Callback Handler Settings Dialog window: The number of fields that are shown in the Callback Handler Settings Dialog window depends on the policy set and policy type for which you are configuring a binding.
 - A. Type the path and name of the keystore file that you received from the web service provider in the **Keystore Path** field, or select the file by using **Browse**.
 - B. Type the password for your keystore in the **Keystore Password** field.
 - C. Select the type of your keystore from the **Keystore Type** list.
 - D. Type the alias for your key in the **Key Alias** field.
 - E. Type the name for your key in the **Key Name** field.
4. Click **OK**.

Creating a policy set attachment on the server side

You can add security to a web service by attaching policy sets to the service. Each attachment specifies an endpoint, a policy set, and a binding.

Before you begin

Important: Applicable edition: Full profile

Create a web service application EAR file that is in your workspace.

About this task

To create a policy set attachment for your web service:

Procedure

1. In the Services view, right-click a service object, and select **Manage Policy Set Attachment**.
2. For each attachment that you want to create, click **Add** and use the Endpoint Definition window to [attach a policy set to an endpoint and specify a binding](#). Each attachment that you create is listed in the **Application** table.
3. Optional: To configure the binding for a policy type on the server, use the administrative console of the server.
4. Click **Finish** to complete the wizard.

Results

A `policyAttachments.xml` is created in the `META-INF` folder of your web service application EAR file.

Parent topic: [Securing web services](#)

Modifying policy set attachments

After you create policy set attachments for your web service or client, you can modify attachment attributes. For example, you can specify different endpoints and policy sets. For policy set attachments on the client side, you can also specify different bindings and binding configurations.

Before you begin

Important: Applicable edition: Full profile

Create policy set attachments for a web service or client in your workspace.

About this task

For a web service and a client of that service, the policy sets and bindings configuration must match for the service to function correctly. Use the information that you obtained from your web service provider when you specify policy sets and most bindings configuration on the client side.

Procedure

1. In the Services view, right-click your service or client; then click **Manage Policy Set Attachment**.
2. To modify the endpoint, policy set, or client-side binding of an attachment:
 - A. Select the attachment in the **Application** table.
 - B. Click **Edit**.
 - C. Use the Endpoint Definition window to [attach a policy set to an endpoint and specify a binding](#).
3. To modify the bindings configuration of a client-side attachment:
 - A. Select the attachment in the **Application** table.
 - B. Select the binding in the **Binding Configurations** table.
 - C. Click **Configure** and use the binding configuration window to [configure a binding](#). The window that opens depends on the type of binding that you select.
4. Click **Finish** to complete the wizard.

Parent topic: [Securing web services](#)

Editing policy set bindings

You can create and edit existing policy set binding configurations by using the binding stand-alone editors.

About this task

Important: Applicable edition: Full profile

Applications (both web service providers and clients) might contain binding configurations that are stored within the application EAR file in the `META-INF` folder for managed applications, and within the `project_src` folder for the thin client project. The policy set binding stand-alone editors abstract these configurations to the level of the service and the client. In the Services view, you can edit or create binding configurations that invoke the editors without knowing the location of these files.

You can use the stand-alone editor to:

- Create a new set of binding configuration and add it to the application
- Edit existing binding configurations that are already within the application

To attach the binding configurations being created or edited to the application endpoints you must use the policy set attachment wizard.

Parent topic: [Securing web services](#)

Creating a binding configuration based on a template Procedure

1. Open the **Services** view.
2. Select the web service provider or client under the JAX-WS tree.
3. Right-click and select **Configure Bindings > Add Binding configuration**. The Create Binding Configuration dialog opens.
 - A. Binding name: Enter the name of the binding set.
 - B. Template: Select from a pre-existing template that is based on the WebSphere Application Server general bindings, or an empty binding skeleton.
 - C. Policy types: Select from the list of all binding policy types that are supported by the server.

Results

A folder with the specified binding name is created, and the binding files from the template are copied into that folder. An editor for each policy type is launched with which you can configure each policy type binding.

Editing an existing binding configuration for an application Procedure

1. Open the **Services** view.
2. Select the web service provider or client under the JAX-WS tree.
3. Right-click and select **Configure Bindings > Edit Binding configuration**. The Configure Bindings dialog opens.

4. You can clear the checkbox next to any policy type to remove it from the binding, or leave all policy sets as they are. When you click **OK**, an editor for each policy type is launched with which you can configure each policy type binding.

Adding a binding configuration to an existing binding based on a template

Procedure

1. Open the **Services** view.
2. Select the web service provider or client under the JAX-WS tree.
3. Right-click and select **Configure Bindings > Add Binding configuration**. The Create Binding Configuration dialog opens.
 - A. Binding name: Enter the name of the binding set you previously created and want to extend.
 - B. Template: Select from a pre-existing template that is based on the WebSphere Application Server general bindings, or an empty binding skeleton.
 - C. Policy types: Select from the list of all binding policy types that are supported by the server. Policy sets that are already configured for this binding are not selected by default. If you want to launch the editor for a policy type, select that policy set. **Note:** Selecting an already configured policy set generates a warning because the previous binding configuration for that policy set type is lost.

Results

The new binding files from the template are copied into the bindings folder. An editor for each policy type is launched with which you can configure each policy type binding.

Importing policy sets into your workspace

In addition to using the policy sets that come with the product, you can import policy sets that were exported from a server. After you import these policy sets into your workspace, you can attach them to your web services and clients.

Before you begin

Important: Applicable edition: Full profile

Your file system contains policy sets that were exported from a server. You might obtain these policy sets, for example, by using an administrative console for the server or by contacting your web service provider. Exported policy sets are stored in .zip files.

About this task

To import policy sets into your workspace:

Procedure

1. On the main menu, click **Import > Web services > WebSphere Policy Sets**.
2. Browse to the .zip file that contains the policy set, and click **Finish**. The policy sets that you imported are displayed in the Service Policies preferences page.

What to do next

After you import a policy set, you can set it as the default, delete it, or restore the original settings in the Service Policies Preferences page.

- If you want to delete a policy set, select the set in the list and click **Delete**. You cannot delete a policy set that comes with the product.
- If you want to restore the original policy set list and the default policy set to the original settings when you installed the product, click **Restore Defaults**.

Parent topic: [Securing web services](#)

Deploying web services

Deploying a web service involves creating the code that makes your web service available to others. You can deploy a project, an EAR file, or an application client. If you created a web service by using the web services wizards, the deployment code is generated automatically.

- **Generating JAX-WS deployment descriptors**

By default, the deployment information for JAX-WS web services that are deployed to WebSphere® Application Server is generated at run time by the server. However, static deployment descriptors for the web service and client can be generated and modified.

- **Deploying a web service to a server using the command line tools**

After you create an EAR file by using a web services command-line tool, you can deploy the EAR file to a server to create the code that makes your service available to others.

- **Creating web service router modules**

The Create Router Modules wizard enables a set of web services within an Enterprise Application Archive (EAR) file. For each web service-enabled EJB JAR file in the EAR file, it adds an HTTP router, a JMS router, or both to the EAR. Each router module provides a web service endpoint for a particular transport. For example, an HTTP router module can be added so that the web service can receive requests over the HTTP transport, and a JMS router module can be added so that the web service can receive requests from a JMS queue or topic. The Create Router Modules wizard was formerly known as the Endpoint Enabler.

Parent topic: [Developing web service applications](#)

Generating JAX-WS deployment descriptors

By default, the deployment information for JAX-WS web services that are deployed to WebSphere® Application Server is generated at run time by the server. However, static deployment descriptors for the web service and client can be generated and modified.

Before you begin

To create a deployment descriptor, you need a JAX-WS web service or client in your workspace.

Procedure

In the Services or Enterprise Explorer views of the Java™ EE perspective, right-click your web service or client, and select **Generate > Deployment Descriptor**. A `webservices.xml` file is generated in the `WebContent/WEB-INF` folder.

Parent topic: [Deploying web services](#)

Deploying a web service to a server using the command line tools

After you create an EAR file by using a web services command-line tool, you can deploy the EAR file to a server to create the code that makes your service available to others.

About this task

You can deploy the EAR file to WebSphere® Application Server by using either the development environment or the administrative console. To deploy the EAR file and test the web service in the development environment, you must import the EAR file then create, configure, and start the web server.

Parent topic: [Deploying web services](#)

Importing an EAR file Procedure

1. Open a workspace.
2. Go to **File > Import > EAR file**, and click **Next**.
3. Click **Browse** to select the command-line test directory from which you ran the web services command-line tool. Click **Open**.
4. Expand your project directory, which was specified by **-project** when you ran the web services command-line tool, and select the EAR file that you created. Click **Open**.
5. In the Import EAR wizard, target the EAR to the appropriate server. Accept the default settings in the next two windows.
Tip: Note the name of the web project that was created with the EAR file. If you are importing an EAR file that contains an EJB bean, also note the name of the EJB project. You need this information later to test your web service.
6. Click **Finish**.

Results

You imported an EAR file and its corresponding project.

You can now create and configure the server to deploy the web service.

Creating and configuring the web server About this task

If you already created a server, go to step 4. Alternately you can create a server manually.

Procedure

1. Select **File > New > Other**. Select **Server** and **Server** from the lists. Click **Next**.
2. Specify the host name of the server you want to publish on, or accept the default localhost setting. Select the type of server you want to create and click **Next**.
3. Select the port that you want to use, and accept the other default settings. Click **Finish**.

4. Switch to the Servers view by going to **Window > Show view > Other > Services**.
5. In the Server Configuration pane, right-click your server. Select **Add and Remove projects**. Select your web service EAR and add it to the server. Click **Finish**.

Results

After you deploy your web service to the server, you can test it.

Testing and validating web services

After you create a web service or client, you can test it using sample JSPs, the web services Explorer, or the Generic Service Client. You can also test the SOAP traffic that is passed by the service.

Testing web services

- [Testing web services using sample JSPs](#)
- [Using the TCP/IP Monitor to test Web services](#)

Validating web services

- [Validating WS-I Web service traffic compliance](#)
- [Validating WSDL](#)

- [Testing web services with the Generic Service Client](#)

The Generic Service Client allows testing of a greater variety of web services than other testing mechanisms - such as JMS web services or secured web services.

- [Testing web services with sample JSPs](#)

After you create a web service, you can generate sample JSPs that can be run on the server to test the web service.

Parent topic: [Developing web service applications](#)

Related tasks:

[Testing WSDL documents and Web services with the WSDL Explorer](#)

[Testing web services with sample JSPs](#)

Testing web services with the Generic Service Client

The Generic Service Client allows testing of a greater variety of web services than other testing mechanisms - such as JMS web services or secured web services.

The generic service client invokes calls to any kind of service that uses an HTTP, JMS, or WebSphere® MQ transport and views the message that is returned by the service. The generic service client is useful for debugging or testing a service when you do not have access to a dedicated client to invoke the service call. You can set up a large variety of transport and security configurations for the service, edit the parameters of the call and send attachments.

For detailed information about how to use the generic service client to test web services, see [Sending service requests with the generic service client](#).

Supported services

The generic service client can invoke requests for many types of services that use the following transport protocols:

- HTTP
- Java™ Message Service (JMS), including JBoss and WebSphere® implementations
- WebSphere MQ

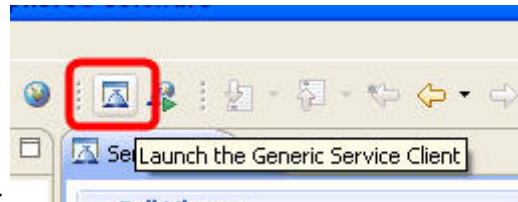
The Generic Service Client does not currently support testing secured web services.

Watch a demonstration of the Generic Service Client being used to test a web service: [Testing web services with the Generic Service Client](#)

Launching the generic service client

You can launch the Generic Service Client in the following ways:

- Right-click a WSDL file in the Enterprise Explorer view, and select **Web services > Test with Generic Service Client**
- Right-click a service node in the Services view, and select **Test with Generic Service Client**
- From the **Run** menu, select **Launch the Generic Service Client**
- In the web services wizards, select the Generic Service Client as your test facility. You can also set it as your default test facility in the web services Preferences.



- Click the **Launch the Generic Service Client** icon in the toolbar:

Invoking the generic service client from a WSDL file

Before you begin, ensure that you have a valid Web Services Description Language (WSDL) file in your workspace (as opposed to a dynamically generated WSDL file as is supported in JAX-WS web services). Ensure that the WSDL files use the correct syntax for the test environment. The generic service client might not work with some WSDL files.

To invoke a service call based on a WSDL file, complete the following steps:

1. Launch the generic service client, and add the WSDL file to it if necessary. The WSDL file is added to the Call Library.
2. In the Call Library, expand the WSDL, binding, and operation, and then select the call element. The call is automatically configured with any SOAP or JMS endpoints that are available in the WSDL. The generic service client shows three steps: **Edit Data**, **Invoke**, and **View Response**. The details for the call are displayed under the **Edit Data** step.
3. On the **Message** page, use the Form, Tree, or Source views to edit the contents of the call. Each view displays the same data differently. If you want to add or remove XML elements in the Form or Tree view, click **Schema > Validate and Assist** to comply with an XML Schema Definition (XSD) specified in the Schema Catalog.
4. On the **Attachments** page, specify any file attachments that you want to be sent with the call. You must first configure the environment with the correct libraries and configuration files to handle file attachments.

5. On the **Transport** page, specify an HTTP, JMS, or WebSphere MQ transport configuration to be used by the call. You can create and edit transport and security configurations by clicking the **Transport** tab.
6. On the **Security for Request** and **Security for Response** pages, specify whether you want to override the security settings for the WSDL. If you want to edit the current security settings for the WSDL, click **Edit WSDL Security** to open the WSDL Security Editor.
7. When you are ready, click **Invoke** to invoke the service call. The generic service client sends the request and displays the message return under the **View Response** step.

Limitations:

Arrays are not supported.

Because of a lack of specification, attachments are not supported with the Java Message Service (JMS) transport. The envelope is directly sent using UTF-8 encoding.

All security algorithms are not always available for every Java Runtime Environment (JRE) implementation. If a particular security implementation is not available, add the required libraries to the class path of the JRE that this product uses.

The generic service tester displays the envelope as reflected in the XML document. However, security algorithms consider the envelope as a binary. Therefore, you must set up the SOAP security configuration so that incoming and outgoing messages are correctly encrypted but remain decrypted inside the test.

Parent topic: [Testing and validating web services](#)

Service testing overview



The service testing capabilities of IBM® Rational® Performance Tester or IBM Rational Service Tester for SOA Quality automate the creation, execution and analysis of functional, regression and performance tests for SOAP-based web services, including support for Java Message Service (JMS), Websphere MQ, and Microsoft .NET Windows Communication Foundation (WCF), as well as any service that produces XML, plain text, or binary data.

Informative test results rely upon sound test development. Each of the following stages contributes to generating meaningful test results:

- **Preparation.** Set up your test environment with the libraries and configuration files required for SOAP-based web services or custom security algorithms. You can import Web Service Description Language (WSDL) definition files and digital certificates that are required by the web services to automatically generate your tests. You can create SOAP security profiles with security algorithms for the web service calls and message returns.
- **Test creation:** Create your test by recording the service requests and responses either with the generic service client, or with an existing client or a web browser through a recording proxy. When you start the recording, you interact with the service by performing service requests and receiving responses. You can also create service tests manually or from a synchronous Business Process Execution Language (BPEL) model.
- **Test editing:** After recording, you can edit the requests and responses in the test. You can use XML Schema Description (XSD) documents to facilitate XML edition. You can replace recorded test values with variable test data, or add dynamic data to the test.
- **Functional testing:** You can run the test to ensure that service matches the expected behavior defined in verification points. During the run, each verification point is checked and receives a pass, fail or inconclusive status.
- **Performance testing:** If you are using IBM Rational Performance Tester, you can specify an execution schedule and user groups to emulate a workload that is generated by a large number of virtual users. Then, you can run the schedule, deploying test execution on virtual users that can be hosted on remote computers. Each virtual user runs an instance of the test client. Response times are measured and recorded. Verification points are checked and recorded.
- **Stub simulation:** Service stubs are functional simulations of an existing service. Service stubs are useful for replacing a service that is unavailable or impractical to use in a test environment. They can also be used to input specific data into a service under test or for prototyping. You can deploy stubs onto a stub server, which can replace the actual server in your test or development environment.
- **Evaluation of results:** You evaluate the results that the tests produce through the performance and verification point reports that are generated during execution. You can also design custom reports by manipulating various counters. Functional reports provide a comprehensive view of the behavior of the service under test. Reports can be exported and archived for validation.

Service testing tools

The following tools are available in the product:

- The generic service client enables you to manually perform service requests for a wide variety of transport protocols, authentication configurations and security profiles, making it an extremely versatile service client. It effectively replaces a dedicated client and can be used to record service calls or for manual testing and debugging a service during development. To open the generic service client, click the **Generic Service Client**  toolbar button.
- The WSDL security editor allows you to set up sophisticated algorithm stacks for your service requests and responses. Algorithm stacks contain digital certificate information and the security algorithms that are applied to messages to perform secure communication with a web service. Algorithm stacks are made of blocks, which can be key definitions, encryption, time stamp, or signature operations which can be associated with any operation in the WSDL file. To open the WSDL security editor, right-click a WSDL file in your workspace and select **Edit WSDL Security** or click the **WSDL Security Editor**  button in the generic service client.
- The test editor is where you develop your test. After recording, you can modify the test to add data correlation or verification points. You can also add loops and conditions and you can edit every detail of the service requests.
- The stub editor enables you to create service stubs. With the stub editor, you can define multiple input conditions, which are similar to verification points. Each condition triggers a predefined simulated response, which is functionally identical to a response from the simulated service.
- In Rational Performance Tester, the schedule editor lets you deploy multiple virtual users on local and remote computers to generate a heavy load for performance testing. A schedule typically contains multiple tests and multiple virtual users.

Related concepts:

[Service testing guidelines](#)

Related tasks:

[Verifying WSDL syntax compliance for JMS services](#)

Recording service tests

When you record a test, the test creation wizard records your interactions with the service, generates a test from the recording, and opens the test for editing. You can record a test session by invoking service calls with the generic service client or by using an existing client. You can also create a service test manually or from a Business Process Execution Language (BPEL) model.

- **Service testing guidelines**

Before you can test a service, you must set up your test environment and incorporate these guidelines in order to produce reliable tests.

- **Verifying WSDL syntax compliance for JMS services**

Various Java™ Message Service (JMS) providers vary in the syntax used for describing services. Before testing JMS services, you must ensure that Web Services Description Language (WSDL) files comply with the requirements of the tool.

- **Recording a service test with the generic service client**

You can record a service test by invoking service requests with the generic service client. After you have sent the requests and received the responses from the service, select the results in the History section of the generic service client to generate a test. If you do not have access to a dedicated client for the service calls, the generic service client is the easiest way to generate the calls and to record a test.

- **Recording a service test through a client program**

You can record tests for SOAP-based, XML, plain text, or binary services with any client program that uses the HTTP protocol. To record the test, the recorder intercepts the service calls and message returns between the client and the service. You can choose between an HTTP or SOCKS proxy recorder or a low-level socket recorder, depending on the capabilities of the client program.

- **Creating a service test from a BPEL model**

You can use Business Process Execution Language (BPEL) resources from your workspace to automatically generate a set of service tests that corresponds to the paths that are run in a synchronous BPEL model.

- **Creating a service test manually**

You can create a service test without recording by simply adding the test elements as required and manually editing the test element details in the test editor.

- **Creating a service test for WebSphere MQ**

You can create an IBM® WebSphere® MQ test by adding the test elements as required and editing the test element details in the test editor.

- **Creating a service test for a plain XML call**

You can create a test for a plain XML call over HTTP, JMS, or IBM WebSphere MQ, by simply adding the test elements as required and editing the test element details in the test editor.

- **Changing service test generation preferences**

You can change default test generation values by changing the preference settings. The default settings, however, are appropriate for recording in most cases.

Service testing guidelines

Before you can test a service, you must set up your test environment and incorporate these guidelines in order to produce reliable tests.

Test prerequisites

Before creating service tests, you might need to perform some initial tasks. These tasks depend on the transport and security protocols that are implemented by the web service under test.

- **HTTP**: This transport method is supported by default; no additional configuration is required.
- **SSL**: The workspace must contain the certificate keystore (JKS) files that are required for single or double authentication.
- **Java™ Message Service (JMS)**: The Web Services Description Language (WSDL) syntax must be compatible with the requirements of the product. Refer to [Verifying WSDL syntax compliance for JMS services](#).

Test generation

When the test is generated, message call envelopes are created according to the XML schema definition (XSD). During this process, mandatory fields are created, and default choices are assumed. You can modify these elements in the test editor.

Note: During recording, you might supply authentication details which are not relevant for the actual application under test. To exclude such actions from the generated test, in **Window > Preferences > Test > Test editor > Service test** ensure that the **Display the 'Skip if Empty' column in XML tree viewer** check box is selected. To select the empty XML elements that you want to skip, in the test editor, select the elements in the **Skip if empty** column.

Encryption and security

The Java Runtime Environment (JRE) that the workbench uses must support the level of encryption required by the digital certificate that you select. For example, you cannot use a digital certificate that requires 256-bit encryption with a JRE that supports only 128-bit encryption. By default, the workbench is configured with restricted or limited strength ciphers. To use less restricted encryption algorithms, you must download and apply the unlimited jurisdiction policy files (`local_policy.jar` and `US_export_policy.jar`).

You can download unlimited jurisdiction policy files from this site:

<http://www.ibm.com/developerworks/java/jdk/security/50/>

Click on **IBM SDK Policy files**, and then log in to developerWorks® to obtain the unlimited jurisdiction policy files. Before installing these policy files, back up the existing policy files in case you want to restore the original files later. Then overwrite the files in `/jre/lib/security/` directory with the unlimited jurisdiction policy files.

SSL Authentication

Service tests support simple or double SSL authentication mechanisms:

- Simple authentication (server authentication): In this case, the test client needs to

determine whether the service can be trusted. You do not need to setup a key store. If you select the **Always trust** option, you do not need to provide a server certificate key store.

If you want to really authenticate the service, you can configure an certificate trust store, which contains the certificates of trusted services. In this case, the test will expect to receive a valid certificate.

- Double authentication (client and server authentication): In this case, the service needs to authenticate the test client according to its root authority. You must provide the client certificate keystore that needs to be produced to authenticate the test as a certified client.

When recording a service test through a proxy, the recording proxy sits between the service and the client. In this case, you must configure the SSL settings of the recording proxy to authenticate itself as the actual service to the client (for simple authentication), and as the client to the service (for double authentication). This means that you must supply the recording proxy with the adequate certificates. When using stub services, you can also configure the SSL settings of the stub service to authenticate itself as the actual server. This means that you must supply the service stub with the adequate certificate.

NTLM and Kerberos Authentication

The product supports Microsoft NT LAN Manager (NTLMv1 and NTLMv2) and Kerberos authentication. The authentication information is recorded as part of the test during the recording phase.

To enable NTLMv2 support, you must add a third party library to the workbench. For more information, see [Configuring the workbench for NTLMv2 authentication](#).

Digital certificates

You can test services with digital certificates for both SSL and SOAP security protocol. Digital certificates must be contained in Java Key Store (JKS) keystore resources that are accessible in the workspace. When dealing with keystore files, you must set the password required to access the keys both in the security editor and the test editor. For SOAP security you might have to provide an explicit name for the key and provide a password to access the private keys in the keystore.

Limitations

Arrays are not supported.

Because of a lack of specification, attachments are not supported with the Java Message Service (JMS) transport. The envelope is directly sent using UTF-8 encoding.

All security algorithms are not always available for every Java Runtime Environment (JRE) implementation. If a particular security implementation is not available, add the required libraries to the class path of the JRE that this product uses.

The generic service tester displays the envelope as reflected in the XML document. However, security algorithms consider the envelope as a binary. Therefore, you must set up the SOAP security configuration so that incoming and outgoing messages are correctly encrypted but remain decrypted inside the test.

Performance

Virtual user performance depends on the implementation of the container application. For an HTTP transport, the product has been tested with a maximum of 900 concurrent virtual users under Windows and 600 under Linux. For JMS, the maximum is 100 concurrent virtual users, although this number can vary due to the asynchronous implementation of JMS. Beyond these values, connection errors might occur and the transaction rate will decrease.

Related concepts:

[Service testing overview](#)

Related tasks:

[Verifying WSDL syntax compliance for JMS services](#)
[Recording a service test with the generic service client](#)
[Recording a service test through a client program](#)
[Creating a service test from a BPEL model](#)
[Creating a service test manually](#)
[Creating a service test for WebSphere MQ](#)
[Creating a service test for a plain XML call](#)
[Changing service test generation preferences](#)

Verifying WSDL syntax compliance for JMS services

Various Java™ Message Service (JMS) providers vary in the syntax used for describing services. Before testing JMS services, you must ensure that Web Services Description Language (WSDL) files comply with the requirements of the tool.

Procedure

1. In the project explorer or test explorer, locate and open the WSDL file for the JMS service that you want to test. If necessary, you can import a WSDL file from the file system by clicking **File > Import > File System**.
2. Ensure that the following criteria are met in the syntax of the WSDL file that you use.
 - Namespace: `xmlns:jms="http://schemas.xmlsoap.org/wsdl/jms/"`
 - SOAP bindings are set to:
`transport="http://schemas.xmlsoap.org/soap/jms"`
 - JMS transports are defined either as a URL or as `jms:address` element
3. If the WSDL file is not compliant, edit the file so that it meets the criteria, and then save and close the file.

Example

For example, a JMS defined as a URL looks like this:`<soap:address`

```
location="jms:/queue?jndiConnectionFactoryName=UJL2ConnectionFactory;
    jndiDestinationName=queue/testQueue;
    initialContextFactory=org.jnp.interfaces.NamingContextFactory;
    jndiProviderURL=9.143.104.47" />
```

A JMS defined as an address looks like this:`<jms:address destinationStyle="queue"`

```
    jndiConnectionFactoryName="myQCF"
    jndiDestinationName="myQ"
    initialContextFactory="com.ibm.NamingFactory"
    jndiProviderURL="iiop://something:900/">
</jms:address>
```

Related concepts:

[Service testing guidelines](#)

[Service testing overview](#)

Related tasks:

[Recording a service test with the generic service client](#)

[Recording a service test through a client program](#)

[Creating a service test from a BPEL model](#)

[Creating a service test manually](#)
[Creating a service test for WebSphere MQ](#)
[Creating a service test for a plain XML call](#)
[Changing service test generation preferences](#)

Recording a service test with the generic service client

You can record a service test by invoking service requests with the generic service client. After you have sent the requests and received the responses from the service, select the results in the History section of the generic service client to generate a test. If you do not have access to a dedicated client for the service calls, the generic service client is the easiest way to generate the calls and to record a test.

Before you begin

If you are testing a SOAP-based web service, ensure that you have access to a valid Web Services Description Language (WSDL) file. The wizard can import WSDL files from the workspace, the file system, a remote repository, or from a URL. Ensure that the WSDL files use the correct syntax for the test environment. The generic service client might not work with some WSDL files.

If you are using Secure Sockets Layer (SSL) authentication, ensure that you have the required key files in your workspace.

If you are using SOAP security, ensure that you have configured the environment with the correct libraries and configuration files. See [Configuring the environment for SOAP security](#) for more information.

If the response in a recording or test generation is in XML and the size of the XML data is higher than the value set in the **XML Message Received maximum length** field, the response is automatically converted to text to avoid any memory issues. To convert the full response to text, the tool checks the value set for **Text Message Received maximum length**. If the value is lesser than the size of the response, the response is truncated. If you want the response to be in XML when the response size exceeds the value set in **XML Message Received maximum length**, you can manually increase the value for both recording and test generation. To change the value for recording, click **Window > Preferences > Generic Service Client > Message Edition**. To change the value for test generation, click **Window > Preferences > Test > Test Generation > Service Test Generation**.

About this task

To use a WS-SecurityPolicy that is included in a WSDL or an external XML file, you need to configure the security policy as described in [Using a security policy](#). If a recording contains the Security Assertion Markup Language (SAML) token, the WS Security policy file must rely on the Service Token Service (STS) that produces the token. This token can then be used for encryption or other purposes as was designed. Sample policy file that relies on SAML token:





```
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
<wsp:Policy>
<sp:IssuedToken sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">
<sp:Issuer>
<Address xmlns="http://www.w3.org/2005/08/addressing">http://9.143.105.204:8080/axis2/services/STS</Address>
</sp:Issuer>
```

```


<sp:RequestSecurityTokenTemplate>
<t:TokenType xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">http://docs.oasis-open.org/wss/oasis-wss-saml-token-
profile-1.1#SAMLV2.0</t:TokenType>
<t:KeyType
xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey</t:KeyType
>
<t:KeySize xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">256</t:KeySize>
</sp:RequestSecurityTokenTemplate>
<wsp:Policy>
<sp:RequireInternalReference/>
</wsp:Policy>
</sp:IssuedToken>
</wsp:Policy>
</sp:SupportingTokens>

```


Procedure

1. In the Performance Test perspective, click the **New Test from Recording** toolbar button  or click **File > New > Test from Recording**.
2. In the New Test from Recording wizard, click **Create a test from a new recording**, select **Service Test**, and click **Next**. If you are recording sensitive data, you can select a **Recording encryption level**.
3. On the Select Location page, select the project and folder where you want to create the test, type a name for the test, and click **Next**. If necessary, click **Create Parent Folder**  to create a project or folder.
4. On the Select Location page, select **Generic Service Client**. This option uses the generic service client if you do not have access to a dedicated client for the service calls. See [Recording a service test through a client program](#) for information about using other client programs to record the test.
5. Click **Next**. If this is the first time you are recording a web service test, read the Privacy Warning, select **Accept**, and click **Finish** to proceed. The generic service client opens.
6. If your service uses a transport or authentication protocol that requires overriding the default settings, then click the **Transport** tab and create an HTTP, Java Message Service (JMS), or IBM® WebSphere® MQ transport.
7. Click the **Requests** tab.
 - Click **Add a WSDL file**  to use a WSDL file from the workspace, to import a WSDL file, or to link to a remote WSDL file.
 - Select the **Add an endpoint**  file to create a call to an HTTP, JMS, or WebSphere MQ service.

See [Sending service requests with the generic service client](#) for more information about using the generic service client.
8. After creating the call, click the **Edit Data** arrow to change the details of the call if necessary.
9. Click the **Invoke** arrow to invoke the service call. If the call was successful, the response is displayed under the **View Response** arrow.

10. To record a test with multiple calls, repeat steps 6 through 9.
11. When you have finished sending service requests, stop the recorder. You can do this by closing the generic service client or by clicking the **Stop** push button  in the Recorder Control view. If you changed the network settings of the client program as described in step 8, you can revert to the default settings before closing the program. The Generate Service Test wizard opens.
12. Click **Finish**.

What to do next

Alternatively, you can use the generic service client to create, edit, and invoke the calls without recording. Successful responses are added to the **Request History** list. You can select calls in the **Request History** list, and click the **Generate Test Suite** icon .

Related concepts:

[Service testing guidelines](#)

[Generic service client overview](#)

Related tasks:

[Verifying WSDL syntax compliance for JMS services](#)

[Recording a service test through a client program](#)

[Creating a service test from a BPEL model](#)

[Creating a service test manually](#)

[Creating a service test for WebSphere MQ](#)

[Creating a service test for a plain XML call](#)

[Changing service test generation preferences](#)

[Sending service requests with the generic service client](#)

[Recording a service test through a client program](#)

[Sending service requests with WSDL files](#)

Recording a service test through a client program

You can record tests for SOAP-based, XML, plain text, or binary services with any client program that uses the HTTP protocol. To record the test, the recorder intercepts the service calls and message returns between the client and the service. You can choose between an HTTP or SOCKS proxy recorder or a low-level socket recorder, depending on the capabilities of the client program.

Before you begin

The following recorders are available for recording traffic from an application:

- SOCKS proxy recorder: Use this recorder when no proxy connections are required.
- HTTP proxy recorder: Use this recorder when a proxy connections is required to connect to the network or when the client program does not support SOCKS.
- Socket recorder: Use this recorder for low-level network traffic when the client does not support proxies. This recorder does not support SSL authentication or encryption of any kind and is only available if the IBM® Rational® Performance Tester Extension *for Socket Protocols* is installed.

Regardless of the recorder that you use, the client program must use the HTTP network protocol. For recording Java™ Message Service (JMS) or IBM WebSphere® MQ tests, see [Recording a service test with the generic service client](#).


If you are using Secure Sockets Layer (SSL), the HTTP or SOCKS proxy can cause authentication problems because the proxy recorder relays traffic between the client and the server. Depending on the authentication method in place, the client might require that the proxy recorder authenticate itself as the server and the server might require that the proxy recorder authenticate as the client. If the client program requires an authenticated server, you must either have access to the server certificate keystore and provide it to the proxy recorder or configure the client to accept the default certificate from the proxy recorder instead of the certificate from the actual server.


If you are testing a SOAP-based web service, ensure that you have access to a valid Web Services Description Language (WSDL) file. The wizard can import WSDL files from the workspace, the file system, a remote repository, or from a URL. Ensure that the WSDL files use the correct syntax for the test environment. The generic service client might not work with some WSDL files.

If you are using SOAP security, ensure that you have configured the environment with the correct libraries and configuration files. See [Configuring the environment for SOAP security](#) for more information.








Procedure

To record a service test with a client program:

1. In the Performance Test perspective, click the **New Test from Recording** toolbar button  or click **File > New > Test from Recording**.
2. In the New Test from Recording wizard, click **Create a test from a new recording**, select **Service Test**, and click **Next**. If you are recording sensitive data, you can select a **Recording encryption level**.

3. On the Select Location page, select the project and folder to create the test in, type a name for the test, and click **Next**. If necessary, click **Create Parent Folder**  to create a project or folder
4. On the Select Client Application page, select the type of client program to use. The program type defines the recorder that can be used. The following client program types are supported for recording a service test:
 - **Managed Application**: This option starts a specified program and uses a proxy or socket recorder to record the traffic. On the Managed Application Options page, click **Browse** to specify the **Program path**. If necessary, specify the **Working directory**, and type the command line **Arguments** that the program requires.
If the program requires user input from a command-line interface, select **Open console for user input**.
 - **Microsoft Internet Explorer or Mozilla Firefox**: This option records traffic that is sent and received with either web browser.
 - **Unmanaged Application**: This option enables you to record traffic from one or multiple client programs that use a proxy. You must manually start the client programs and the proxy recorder records all traffic that is sent and received through the specified network port.
 - **Generic Service Client**: This option uses the generic service client if you do not have access to a dedicated client for the service calls. See [Recording a service test with the generic service client](#) for using the generic service client to record service tests.
5. On the Recorder Settings page, depending on the type of client program you selected, specify these details:
 - A. If you selected **Managed Application**, specify the recording method.
 - Select **Record traffic with the proxy recorder** to record HTTP or SOCKS traffic through a proxy.
 - Select **Record traffic with the socket recorder** to record low-level network traffic for applications where a proxy cannot be used. This recorder does not support SSL authentication or encryption.

Note: When using proxy recording, you can filter out HTTP or HTTPS requests to a specific endpoints so that any requests to those endpoints are not recorded.
 - B. If you selected **Record traffic with the proxy recorder**, specify whether the proxy recorder uses HTTP or SOCKS. Select **HTTP** if a connection to proxy is required or if your application does not support SOCKS.
 - C. If you are using SSL authentication, specify the authentication settings for the proxy recorder. During the recording, the proxy recorder is between the client and the server.
 - If the server requires client authentication, you must provide the client certificate keystore for the proxy recorder to be authenticated by the server as though the proxy recorder were the client. Select **The server requires a specific client certificate**. Specify the filename and password of the server certificate keystore. If multiple certificates are required, click **Multiple certificates**, and click **Add** to specify a certificate keystore file name and password for each host name and port.**Note:** The keystore must contain the private certificate of the client.

- If the client requires server authentication, you must provide the server certificate keystore for the proxy recorder to be authenticated by the client as though the proxy recorder were the server. Select **The client requires a specific server certificate**, and click **Add** to specify a certificate keystore filename and password for each hostname and port. If you do not select this option, the proxy recorder provides its own default certificate. **Note:** The keystore must contain the private certificate of the server.
- D. If you selected to use the HTTP proxy recorder, specify how to connect to the network. If necessary, specify an HTTP or SOCKS proxy or point to a proxy auto-configuration (PAC) file. Use this option if you are connecting to the service through a corporate proxy or firewall.
6. Click **Next**. If this is the first time you record a service test and you did not select a web browser for the client application, read the Privacy Warning, select **Accept**, and click **Finish** to proceed.
 7. If you selected a proxy recorder with a managed or unmanaged application, change the network settings of the client program to use the proxy recorder. The method for changing the network settings depends on the client program. However, you must be able to set the following proxy settings in the program:
 - SOCKS or HTTP proxy: Specify the protocol that you selected for the proxy recorder in the wizard.
 - Host name: Set to `localhost`.
 - Port: Specify the port number that you selected for the proxy recorder in the wizard.
 To avoid unexpected results, revert to the previous proxy settings before you stop the recording.
 8. Use the client program to perform the actions to test. You can use the Recorder Test Annotations toolbar to add comments, record synchronizations, or take screen captures during the recording.
 - To add a comment to the recorded test, click the **Insert comment** icon .
 - To add a screen capture to the recorded test, click the **Capture screen** icon . Screen and window captures make your tests easier to read and help you visualize the recorded test. You can change the settings for screen captures and add a comment to the image.
 - To manually add a synchronization point to the recording, click the **Insert synchronization** icon .
 - To manually add a transaction folder to the recording, click the **Start Transaction** icon  and **Stop Transaction** icon  to start and stop the transaction.
 - To insert a split point into the recorded test, click the **Split point** icon . With split points, you can generate multiple tests from a single recording, which you can replay in a different order with a schedule.
 9. After you finish the user tasks in the client program, stop the recorder. You can do this by closing the client program or by clicking the button **Stop**  in the Recorder Control view. If you changed the network settings of the client program as described in step 8, you can revert to the default settings before closing the program. The Generate Service Test wizard opens.
 10. If you inserted a split point during the recording, on the Destination page, specify the location for the split test or merge the split recordings together.

11. On the Service Test Generation Options page, if you are testing a SOAP-based web service, specify a Web Services Description Language (WSDL) file from the workspace or click **Add** to import a WSDL or to link to a remote WSDL file.
12. Click **Finish**.

Results

A progress window opens while the test is generated. On completion, the Recorder Control view displays the `Test generation completed` message, the test navigator lists your test, and the test opens in the test editor.

Related concepts:

[Service testing guidelines](#)

Related tasks:

[Verifying WSDL syntax compliance for JMS services](#)
[Recording a service test with the generic service client](#)
[Creating a service test from a BPEL model](#)
[Creating a service test manually](#)
[Creating a service test for WebSphere MQ](#)
[Creating a service test for a plain XML call](#)
[Changing service test generation preferences](#)
[Recording a service test with the generic service client](#)
[Sending service requests with the generic service client](#)
[Sending service requests with WSDL files](#)

Creating a service test from a BPEL model

You can use Business Process Execution Language (BPEL) resources from your workspace to automatically generate a set of service tests that corresponds to the paths that are run in a synchronous BPEL model.

Before you begin

Tests are stored in test projects. If your workspace does not contain a test project, the test creation wizard creates one, enabling you to change its name. To store a test in a specific project, verify that the project exists before you record the test. If you are using Secure Sockets Layer (SSL) authentication, ensure that you have any required key files in your workspace.

If you are using Java™ Message Service (JMS), ensure that you have configured the environment with the correct libraries and configuration files. Ensure that the WSDL files use the correct syntax for the test environment.

If you are using SOAP security, ensure that you have configured the environment with the correct libraries and configuration files.

BPEL models must be synchronous. Asynchronous BPEL models are not supported.

Ensure that the BPEL models refer to the WSDL files in a valid import statement, for example:

```
<bpws:import importType="http://www.w3.org/2001/XMLSchema" location="foo.wsdl" namespace="http://foo"/>
```

Relative file paths, such as: `"../..../foo.wsdl"` are not supported.

Ensure that you have one or more valid Web Services Description Language (WSDL) files and the associated BPEL model in your workspace. Only the calls to services with a valid web service binding are taken into account. For example, if the BPEL model was produced in IBM® Websphere Integration Developer, then services must be exported with the following web service bindings:

```
<bpws:invoke
```

```
name="myOperation" operation="myOperation" partnerLink="IServicePartner" portType="ns3:IService"
```

```
wpc:displayName="myOperation" wpc:id="20">
```

Only BPEL invoke activities are considered for generating tests. Any BPEL receive and reply activities are ignored.

Websphere Integration Developer does not generate the required `soapAction` attributes for the soap operations in the WSDL files. Please edit the generated WSDL files, as follows for every operation: `<soap:operation soapAction=""/>`.

Procedure

To create a service test from a BPEL model:

1. In the Performance Test perspective, click **File > New > Other > Test > Test Assets > BPEL to Web service test**, and then click **Next**.
2. Click **Browse** to select a BPEL file from the workspace, and click **Next**.
3. On the Web service test generation page, change the number of paths by specifying how activities and sequences from the flow of the BPEL model are processed. Each path generates one test.
 - A. In the **Flow** section, select how any concurrent sequences that are found in the flow will be converted into paths.

- B. In the **Switch** section, select whether to test otherwise activities from the flow.
 - C. In the **Throw** section, select how throw activities from the flow are converted into paths.
 - D. In the **Invoke** section, select whether to test inline catches inside invoke activities from the flow.
 - E. Select **Enable data correlation in generated tests** to automatically create references in the generated test elements by propagating variables to the parameters of the web service call and message return elements.
4. Click **Recount paths** to update the number of paths to test, and click **Next**. One test is generated for each path.
 5. For WSDL operations that are bound to multiple ports, you must select one port that is to be used for the test. Under each test that will be generated, the **Operations** list displays the WSDL operations that are bound to multiple ports. If no WSDL operations are displayed under the tests, this means that all operations are bound to a single port. In this case, skip step 6.
 - A. In the **Operations** list, expand a test and select a WSDL operation that requires binding.
 - B. In the **Binding ports** list, select the port that you want to use to test the selected WSDL operation.
 - C. Repeat steps a and b for each WSDL operation that requires binding.
 6. Click **Next**.
 7. Select a location and a name for the new folder where the tests generated from the BPEL model are created, and click **Finish**.

Results

A new folder is created in the Test Navigator containing the generated service tests. These tests are generated with default message content and must be edited with valid input values.

Related concepts:

[Service testing guidelines](#)

Related tasks:

[Verifying WSDL syntax compliance for JMS services](#)

[Recording a service test with the generic service client](#)

[Recording a service test through a client program](#)

[Creating a service test manually](#)

[Creating a service test for WebSphere MQ](#)

[Creating a service test for a plain XML call](#)

[Changing service test generation preferences](#)

Creating a service test manually

You can create a service test without recording by simply adding the test elements as required and manually editing the test element details in the test editor.

Before you begin

Tests are stored in test projects, which are test projects that include a source folder. You must create a test project before creating a test.

Ensure that you have a valid WSDL file in your workspace. Ensure that the WSDL files use the proper syntax for the test environment.

If you are using Secure Sockets Layer (SSL) authentication, ensure that you have any required key files in your workspace.

If you are using SOAP security, ensure that you have configured the environment with the proper libraries and configuration files.

Procedure

1. In the workbench, click **File > New > Other > Test > Test Assets > Service Test** or click the **New Service Test** toolbar button.
2. Select a project and then, in **Test file name**, type a name for the test. The name that you type is the base name for the recording, test, and other required files. You see these files in standard Navigator or the Java™ Package Explorer with their distinguishing suffixes, but you see only the simple (test) name in the Test Navigator.
3. Click **Finish**.
4. To add a web service call, select the test element in the test editor, click **Add**, and then select **Web Service Request**.
5. Select the WSDL file that corresponds to the call and click **Finish**.
6. Edit the web service request element and add all the required information to make a valid call.
7. Select the **Protocol tab** to configure the transport protocol for this call. If necessary, click **Change** to configure the transport protocol for the entire test, including proxy and HTTPS parameters.
8. On the web service call, click **Update Return**. This opens the Return Preview window, displaying the data that will be used to perform the call.
9. Click **Update Test**. This performs the web service call and creates a message return element with the return data. If a message return element already exists, then it is updated with latest return data. The message return test element enables you to implement data correlation and content-based verification points.

Related concepts:

[Service testing guidelines](#)

Related tasks:

[Verifying WSDL syntax compliance for JMS services](#)

[Recording a service test with the generic service client](#)

Recording a service test through a client program
Creating a service test from a BPEL model
Creating a service test for WebSphere MQ
Creating a service test for a plain XML call
Changing service test generation preferences

Creating a service test for WebSphere MQ

You can create an IBM® WebSphere® MQ test by adding the test elements as required and editing the test element details in the test editor.

Before you begin

Tests are stored in test projects, which are Java™ projects that include a source folder. You must create a test project before creating a test.

Ensure that you have a valid Web Services Description Language (WSDL) file for a WebSphere MQ service in your workspace.

If you are using Secure Sockets Layer (SSL) authentication, ensure that you have any required key files in your workspace.

If you are using SOAP security, ensure that you have configured the environment with the correct libraries and configuration files.

Procedure

1. In the workbench, click **File > New > Other > Test > Test Assets > Service Test** or click the **New Service Test** toolbar button.
2. Select a project, and then, in **Test file name**, type a name for the test. The name that you type is the base name for the recording, test, and other required files. You see these files in the standard Navigator or the Java Package Explorer with their distinguishing suffixes, but you see only the simple (test) name in the Test Navigator.
3. Click **Finish**.
4. To add a service call, select the test element in the test editor, click **Add**, and then select **Web Service Call**.
5. Select the WSDL file that corresponds to the call, and click **Finish**.
6. Select the **Protocol tab** to configure the transport protocol for this call. On the Protocol page of the new call element, the MQ protocol is automatically generated with the values from the WSDL file. If necessary, click **Change** to configure the transport protocol for the entire test, including proxy and WebSphere MQ parameters.
 - If the WebSphere MQ server is running on the local computer, select **Using local queue manager**.
 - If the server is on a remote computer, clear **Using local queue manager**, and type the IP address or hostname, port, and client channel.
7. On the web service call, click **Update Return**. This opens the Return Preview window, displaying the data that will be used to perform the call.
8. Click **Update Test**. This performs the web service call and creates a message return element with the return data. If a message return element already exists, then it is updated with latest return data. With the message return test element, you can implement data correlation and content-based verification points.

Related concepts:

[Service testing guidelines](#)

Related tasks:

- Verifying WSDL syntax compliance for JMS services
- Recording a service test with the generic service client
- Recording a service test through a client program
- Creating a service test from a BPEL model
- Creating a service test manually
- Creating a service test for a plain XML call
- Changing service test generation preferences

Creating a service test for a plain XML call

You can create a test for a plain XML call over HTTP, JMS, or IBM® WebSphere® MQ, by simply adding the test elements as required and editing the test element details in the test editor.


Before you begin

Tests are stored in test projects, which are Java™ projects that include a source folder. You must create a test project before creating a test.

If you are using Secure Sockets Layer (SSL) authentication, ensure that you have any required key files in your workspace.

If you are using SOAP security, ensure that you have configured the environment with the correct libraries and configuration files.

Procedure

1. In the workbench, click **File > New > Other > Test > Test Assets > Service Test** or click the **New Service Test**  toolbar button.
2. Select a project, and then, in **Test file name**, type a name for the test and click **Next**. The name that you type is the base name for the recording, test, and other required files. You see these files in the standard Navigator or the Java Package Explorer with their distinguishing suffixes, but you see only the simple (test) name in the Test Navigator.
3. On the Select Service Call Interface page, select whether you want to create a test using a plain **XML call** interface or a **Web service call** interface. If you select web service call interface, select or add a WSDL file and then, select port to which the call will be binded. Click **Next**.
4. On the Configure Protocol page, select either **HTTP**, **JMS** or WebSphere **MQ** as the protocol and then, specify the options for the selected **Protocol configuration**.
5. On the Select Root Element page, you can select an XSD and then, select a root element for the call.
6. Click **Finish**.

Related concepts:

[Service testing guidelines](#)

Related tasks:

[Verifying WSDL syntax compliance for JMS services](#)

[Recording a service test with the generic service client](#)

[Recording a service test through a client program](#)

[Creating a service test from a BPEL model](#)

[Creating a service test manually](#)

[Creating a service test for WebSphere MQ](#)

[Changing service test generation preferences](#)

Changing service test generation preferences

You can change default test generation values by changing the preference settings. The default settings, however, are appropriate for recording in most cases.

Procedure

1. Click **Window > Preferences > Test > Web Services Test Generation**
2. Select the setting to change.
 - **Time out delay used for call**
 - This is the default time out for web service calls. If the web service does not respond within this period, an error is produced.
 - **Think time default value**
 - This is the default think time for generated tests.
3. After changing a setting, click **Apply**.

Related concepts:

[Service testing guidelines](#)

Related tasks:

[Verifying WSDL syntax compliance for JMS services](#)
[Recording a service test with the generic service client](#)
[Recording a service test through a client program](#)
[Creating a service test from a BPEL model](#)
[Creating a service test manually](#)
[Creating a service test for WebSphere MQ](#)
[Creating a service test for a plain XML call](#)

Editing service tests

After you record a service test, you can edit the calls and message returns to include variable data (rather than the data that you recorded). You can add verification points (to confirm that the test runs as expected), transactions, conditional processing, and custom code.

- **Web service test editor overview**

With the test editor, you can inspect or customize a test that you recorded.

- **Verifying application behavior**

To check the expected behavior of the application during a service test, you can add verification points after a message return. During the run, verification points produce a pass, fail, error, or inconclusive status in the Web Service Verification Point report.

- **Adding elements to a service test**

- **Editing WSDL security profiles**

To ensure that your service test uses the correct security protocols to access a SOAP-based service, you must specify a security profile for the (Web Service Description Language) WSDL file. After a security profile is set up, it can be reused in multiple web service calls.

- **Testing asynchronous services**

- **Creating a reliable messaging call structure**

You can create a test structure dedicated to testing service calls based on the WS-ReliableMessaging specification.

Web service test editor overview

With the test editor, you can inspect or customize a test that you recorded.

The test editor lists the web service call elements for a test, in sequential order.

There are two main areas in the test editor window. The area on the left, **Test Contents**, displays the chronological sequence of test elements in the test. The area on the right, **Test Element Details**, displays details about the currently selected item (test, call, message return, or verification point) in the test hierarchy. Window events are the primary test elements in a Citrix test and represent graphic objects that are drawn by the Citrix server, such as actual window, dialog boxes, menus, or tooltips. A Window event is recorded each time a window is created, destroyed, moved, or resized. The first occurrence of a window, a create window event, is displayed in bold. Window objects are typically identified by their title. If there is no window title, for example on menus or tooltips, then the test editor uses the window ID number.

A service request node name can be updated automatically or you can use custom code or datapool to supply different names. To apply a datapool to a node name, in the test editor, select the node name. In the Request Details area of the test editor, clear the **Update node name automatically**, select the name and substitute it with datapool.

Web service calls can contain web service message return elements, which display the results of the web service call. The XML message content can be displayed either in Form, Tree or Source view. Each of these views displays the same message content in different forms:

- **Form** view provides a simplified view of the call elements focused on editing the values of the XML message content.
- **Tree** view provides a hierarchical view of the XML structure, including elements, namespaces, and the associated values. Tree view also allows you to manipulate XML fragments.
- **Source** view displays the XML contents of a web service or XML call or the plain text contents of a simple text message.

Message return elements can contain verification point elements that check that the actual return results match expected criteria.

Some actions contain data that is highlighted. This highlighting indicates that the data can be used as a datapool candidate or as a reference.

In service calls and message returns, you can use datapools and data correlation on values contained in the XML or on XML fragments. To use data correlation on XML fragments, switch to the Tree view, right-click the XML element and select **Create XML Fragment**.

To view or modify the color coding in web service tests, click **Window > Preferences > Test > Test Editor**, and then click the **Fonts and Colors** tab.

Click **Add** to add elements to the selected test element. Alternatively, you can right-click a test element and select an action from a menu.

The choices that you see depend on what you have selected. For example, inside a web service call, you can add a web service message return. The **Insert** button works similarly. Use it to insert an element before the selected element. The

Remove button allows you to delete an item.

Related tasks:

[Verifying application behavior](#)

[Editing WSDL security profiles](#)

[Creating a reliable messaging call structure](#)

[Adding elements to a service test](#)

[Comparing service test response contents](#)

Related reference:

[Service test details](#)

[Service test editor preferences](#)

Related information:

[Adding elements to a service test](#)

[Testing asynchronous services](#)

Verifying application behavior

To check the expected behavior of the application during a service test, you can add verification points after a message return. During the run, verification points produce a pass, fail, error, or inconclusive status in the Web Service Verification Point report.

- **Adding equal verification points**

Equal verification points enable you to check that the contents returned by a service match exactly the contents specified in the verification point.

- **Adding contain verification points**

With contain verification points, you can check that one or several elements of the XML content returned by a service match the XML fragment that is specified in the verification point.

- **Adding Xpath query verification points**

With service query verification points, you can check that a response matches an Xpath query.

- **Adding attachment verification points**

Service attachment verification points enable you to check that the attachment of a service response matches the specified criteria.

- **Adding XSD verification points**

XSD verification points enable you to check that the XML content of a service response comply with the rules defined in an XML Schema Definition (XSD) file.

Related concepts:

[Web service test editor overview](#)

Related tasks:

[Editing WSDL security profiles](#)

[Creating a reliable messaging call structure](#)

[Adding attachment verification points](#)

[Adding equal verification points](#)

[Adding Xpath query verification points](#)

Related information:

[Adding elements to a service test](#)

[Testing asynchronous services](#)

Adding equal verification points

Equal verification points enable you to check that the contents returned by a service match exactly the contents specified in the verification point.

About this task

When you add verification points, the results from a service response are compared with the expected data specified in the verification point test element. Equal or contain verification points enable you to directly compare the XML document that the service returns.

- Contain verification points return a Pass status when the response XML document contains the expected XML data.
- Equal verification points return a Pass status when the response XML document matches exactly the expected XML data.

Complex service requests or verification points might have empty XML elements that are not needed in a test script. When playing back the test, you can skip such empty XML elements. In **Window > Preferences > Test > Test editor > Service test** ensure that the **Display the 'Skip if empty' column in XML tree viewer** check box is selected. This option displays a **Skip if empty** column in the tree view of the request. You can then choose the XML elements to skip.

Procedure

1. Open the test editor, and right click a response element and select **Add > Equal Verification Point**.
2. Select the verification point, and in the **Test Element Details** area of the test editor, type a name for the verification point.
3. Select the verification options:
 - Select **Test using XML namespaces** to perform the verification on the qualified structure of the XML document, including the namespace tagging, instead of the simple name. Disable this option to check only the simple name of the element and the final return value.
 - Select **Text XML text nodes** to include the content of text elements in the verification.
 - Select **Text XML attributes** to include the content of attributes in the verification.
4. On the Message page, select the **Form**, **Tree**, or **Source** view to specify the expected XML data. For an equal verification point, the expected XML data contains the XML document from the response test element. If necessary, you can edit the expected XML data.

You can specify standard Java™ regular expressions in the **Tree** view. To do this, select the **Regular expression** column on the line of an attribute or text value and type the regular expression in the **Value** column. For example, the following regular expression checks for a correctly formatted email address: `/^[a-zA-Z0-9_\. \-]+\@(([a-zA-Z0-9 \-])+\.)+([a-zA-Z0-9]{2,4})+$/`

When using regular expressions, the number of XML nodes or XML fragments in the verification point must match the quantity of expected nodes.

What to do next

You can enable or disable each verification point by right-clicking the verification point in the test editor and clicking **Enable** or **Disable**.

Related tasks:

[Adding contain verification points](#)

[Adding Xpath query verification points](#)

[Adding attachment verification points](#)

[Adding XSD verification points](#)

[Verifying application behavior](#)

Adding contain verification points

With contain verification points, you can check that one or several elements of the XML content returned by a service match the XML fragment that is specified in the verification point.

About this task

When you add verification points, the results from a service response are compared with the expected content that is specified in the verification point test element. Equal or contain verification points enable you to directly compare the XML contents that the service returns.

- Contain verification points return a Pass status when the response XML contents contain the expected XML fragment.
- Equal verification points return a Pass status when the response XML contents match exactly the entire expected XML content.

Complex service requests or verification points might have empty XML elements that are not needed in a test script. When playing back the test, you can skip such empty XML elements. In **Window > Preferences > Test > Test editor > Service test** ensure that the **Display the 'Skip if empty' column in XML tree viewer** check box is selected. This option displays a **Skip if empty** column in the tree view of the request. You can then choose the XML elements to skip.

Procedure

1. Open the test editor, and select a service response element.
2. In the Test Element Details area, click the **Message** tab and select the **Form** or **Tree** view.
3. Expand the envelope line, right click the element that you want to check, and then click **Create Contain Verification Point**. This action creates a contain verification point that includes the XML element from the recorded response.**Note:** You can also create a contain verification point with the message response by selecting the message response in the Test Contents pane and clicking **Add > Contain Verification Point**. However, the result is effectively the same as an equal verification point because the verification point contains the entire XML content of the message response.
4. Select the verification point, and in the Test Element Details pane, type a name for the verification point.
5. Select the verification options:
 - Select the **Test using XML namespaces** check box to perform the verification on the qualified structure of the XML document, including the namespace tagging, instead of the simple name. Disable this option to check only the simple name of the element and the final return value.
 - Select the **Test XML text nodes** check box to include the content of text elements in the verification.
 - Select the **Test XML attributes** check box to include the content of attributes in the verification.

6. If necessary, select the **Form**, **Tree**, or **Source** views to edit the expected XML fragment. For an equal verification point, the expected XML data contains the XML document from the response test element. If necessary, you can edit the expected XML data.

You can specify standard Java™ regular expressions in the **Tree** view. Select the **Regular expression** column on the line of an attribute or text value and type the regular expression in the **Value** column. For example, the following regular expression checks for a correctly formatted email address: `/^([a-zA-Z0-9_\. \-])+\@(([a-zA-Z0-9 \-])+\.)+([a-zA-Z0-9]{2,4})+$/`

When using regular expressions, the number of XML nodes or XML fragments in the verification point must match the number of expected nodes. The verification point returns a Pass status when all regular expressions in the XML fragment are matched.

Example

You can use a contain verification point to check that the message response contains only a specific element with a specific value. For example, consider the following message response:

```
<s:Envelope
xmlns:a="http://www.w3.org/2005/08/addressing"
xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Header>
    <a:Action
      s:mustUnderstand="1">http://www.w3.org/2005/08/addressing/soap/fault</a:Action>
    <a:RelatesTo>uuid:ed9bc447-d739-452f-989d-cd48344d494a</a:RelatesTo>
  </s:Header>
  <s:Body>
    <s:Fault>
      <s:Code>
        <s:Value>s:Sender</s:Value>
        <s:Subcode>
          <s:Value
            xmlns:a="http://schemas.xmlsoap.org/ws/2005/02/sc">a:BadContextToken</s:Value>
          </s:Subcode>
        </s:Code>
        <s:Reason>
          <s:Text
            xml:lang="en-US">The message could not be processed. This is most likely because the action
            &apos;http://Samples.ICalculator/Add&apos; is incorrect or because the message contains an invalid or expired security
            context token or because there is a mismatch between bindings. The security context token would be invalid if the service
            aborted the channel due to inactivity. To prevent the service from aborting idle sessions prematurely increase the
            Receive timeout on the service endpoint&apos;s binding.</s:Text>
          </s:Reason>
          <s:Node>http://www.w3.org/1999/xlink</s:Node>
          <s:Role>http://www.w3.org/1999/xlink</s:Role>
          <s:Detail
            xmlns:tns0="http://schemas.com/2003/10/Serialization/"
            xmlns:tns15="http://Samples.Windows"
            tns0:Id="id"
```



```

tns0:Ref="idref">
  <tns15:GetCallerIdentityResponse>
    <tns15:GetCallerIdentityResult>str</tns15:GetCallerIdentityResult>
  </tns15:GetCallerIdentityResponse>
</s:Detail>
</s:Fault>
</s:Body>
</s:Envelope>

```

To check for the `Subcode` element, the expected content of the contain verification point is the following XML fragment:

```

<s:Subcode
  xmlns:a="http://www.w3.org/2005/08/addressing"
  xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Value
    xmlns:a="http://schemas.xmlsoap.org/ws/2005/02/sc">a:BadContextToken</s:Value>
</s:Subcode>

```

By default, the contain verification point checks whether an element named `Subcode` contains one element named `Value`. You can use the following options:

- **Test using XML namespaces:** With this option, the verification point checks whether an element named `"http://www.w3.org/2003/05/soap-envelope":SubCode` contains one element named `"http://www.w3.org/2003/05/soap-envelope":Value`.
- **Test XML text node:** With this option, the verification point also checks whether the element named `Value` contains the text `a:BadContextToken`.
- **Test XML attributes:** With this option, the verification point also checks that the attributes match the expected XML fragment. In this example, the **Test XML attributes** option is not necessary because the `Subcode` element does not have any attributes.

To check that the `Detail` element properly returns a specific value for `GetCallerIdentityResult`, the expected content of the contain verification point is the following XML fragment:

```

<s:Detail
  xmlns:a="http://www.w3.org/2005/08/addressing"
  xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:tns0="http://schemas.com/2003/10/Serialization/"
  xmlns:tns15="http://Samples.Windows"
  tns0:Id="regular_expression"
  tns0:Ref="idref">
  <tns15:GetCallerIdentityResponse>
    <tns15:GetCallerIdentityResult>IdentityValue</tns15:GetCallerIdentityResult>
  </tns15:GetCallerIdentityResponse>
</s:Detail>

```

You can use the following options:

- **Test XML text node:** With this option, the verification point also checks whether the element named `GetCallerIdentityResult` contains the text `IdentityValue`.
- **Test XML attributes:** With this option, the verification point also checks that the attribute `Id` referred to by `tns0:Id` has the expected value. You can specify a

regular expression for this value by using the **Regular expression** column in the Tree view of the verification point. For example, `tns0:Id="[a-zA-Z]"` checks that the value does not contains numbers.

What to do next

You can enable or disable each verification point by right-clicking the verification point in the test editor and clicking **Enable** or **Disable**.

Related tasks:

- [Adding equal verification points](#)
- [Adding Xpath query verification points](#)
- [Adding attachment verification points](#)
- [Adding XSD verification points](#)

Adding Xpath query verification points

With service query verification points, you can check that a response matches an Xpath query.

Before you begin

When you add verification points, the results from a service response are compared with the expected data that is specified in the verification point test element. With query verification points, you can check that the number of nodes returned by an XML Path language query matches the expected number of nodes specified in the verification point.

Refer to the XPath specification for details on expressing an XPath query:

<http://www.w3.org/TR/xpath>.

You can use the test editor to create or edit verification points.

Procedure

1. Open the test editor, and select a web service response element.
2. Click **Add**, and select **Query verification point**.
3. In the **Test Element Details** area of the test editor, type a name for the verification point.
4. Type a valid **XPath expression** or click **Build Expression** to open the XPath Expression Builder. The XPath Expression Builder helps you build and evaluate XPath expressions based on the recorded contents of the response.
5. Specify a **Comparison operator** (**=**, **>**, or **<**), and the expected number of nodes that the query should return. Click **Evaluate** to update the Expected Count with the actual result based on the recorded contents of the response.

What to do next

You can enable or disable each verification point by right-clicking the verification point in the test editor and clicking **Enable** or **Disable**. **Note:** Because XPath expressions require that the qualified name have a prefix, XPath expressions will return null for the default namespace declared with xmlns.

Related tasks:

[Adding equal verification points](#)

[Adding contain verification points](#)

[Adding attachment verification points](#)

[Adding XSD verification points](#)

[Verifying application behavior](#)

Adding attachment verification points

Service attachment verification points enable you to check that the attachment of a service response matches the specified criteria.

Before you begin

When you add verification points, the results from a service response are compared with the expected data that are specified in the verification point test element.

Attachment verification points enable you to verify that an expected attachment is delivered with the response.

Attachment verification points return a Pass status when all the criteria of an attachment match the expected criteria specified in the verification point test element. If any of the criteria do not match, the verification point returns a Fail status.

You can use the test editor to create or edit verification points.

Procedure

To add attachment verification points to a performance test:

1. Open the test editor and select a service response element.
2. Click **Add** and select **Attachment Verification Point**.
3. In the **Test Element Details** area of the test editor, type a name for the verification point, and specify the criteria to be verified. All criteria must match in order for the verification point to pass.
 - A. In the case of multiple attachments, set the **Index of attachments** to the index number of the attachment to be checked. Type 1 if there is only one attachment in the response.
 - B. Specify the expected size in bytes of the attachment.
 - C. Specify the MIME type and encoding of the attachment.

What to do next

You can enable or disable each verification point by clicking **Enable verification point** in the test editor.

Related tasks:

[Adding equal verification points](#)

[Adding contain verification points](#)

[Adding Xpath query verification points](#)

[Adding XSD verification points](#)

[Verifying application behavior](#)

Adding XSD verification points

XSD verification points enable you to check that the XML content of a service response comply with the rules defined in an XML Schema Definition (XSD) file.

Before you begin

When you add verification points, the results from a service response are compared with the expected data that are specified in the verification point test element. XSD verification points return a Pass status when the XML contents of the response are compliant with the associated XSD or a Web Service Description Language (WSDL) file that contains XSD information.

If you add multiple XSD files to the verification, then the XML content of the response must comply with all of the XSD files.

You can use the test editor to create or edit verification points.

Procedure

To add an XSD verification point to a test:

1. Open the test editor and select a service response element.
2. Click **Add** and select **XSD Verification Point**.
3. In the **Test Element Details** area of the test editor, type a name for the verification point.
4. Click **Add XSD** to add a an XSD file to the validation list or **Add WSDL** to add a WSDL that contains XSD information. Click **Open** to display the XSD or WSDL contents.

What to do next

You can enable or disable each verification point by right-clicking the verification point in the test editor and clicking **Enable** or **Disable**.

Related tasks:

[Adding equal verification points](#)

[Adding contain verification points](#)

[Adding Xpath query verification points](#)

[Adding attachment verification points](#)

Adding elements to a service test

- **Adding a service request**

You can use service request elements in tests to send a request to the service.

- **Updating a service response from the service**

While you are developing a service test, you can send a request from the test editor to record or update the response element.

- **Manually adding a response element**

You can add service response elements to specify the received content of a service request. You can use the test editor to create or edit response elements in an existing service test.

Related concepts:

[Web service test editor overview](#)

Related tasks:

[Verifying application behavior](#)

[Editing WSDL security profiles](#)

[Creating a reliable messaging call structure](#)

Related information:

[Testing asynchronous services](#)

Adding a service request

You can use service request elements in tests to send a request to the service.

About this task

Complex service requests or verification points might have empty XML elements that are not needed in a test script. When playing back the test, you can skip such empty XML elements. In **Window > Preferences > Test > Test editor > Service test** ensure that the **Display the 'Skip if empty' column in XML tree viewer** check box is selected. This option displays a **Skip if empty** column in the tree view of the request. You can then choose the XML elements to skip.

Procedure

1. Open the test in the test editor, and select the first element in the test.
2. Click **Add** and select a service request.
3. If you selected WSDL service request, select one or several WSDL files in your workspace for the web service that you want to test and click **Next**. If necessary, you can import a WSDL file into the workspace with the **Add** button.
4. Select either **HTTP**, **JMS**, or **WebSphere MQ** depending on the transport protocol used by the web service, and provide the correct transport protocol configuration to perform the call. You can create a **New** transport configuration or reuse an existing one.
5. Click **Finish**. This creates the web service request in the test editor.
6. On the **Message** page of the request, select the **Form**, **Tree**, or **Source** views to edit the service request contents.
7. If any resource files are to be attached to the request, select the **Attachment** tab. Use **Add**, **Remove**, or **Edit** to specify the resources that are to be attached to the request.
8. If the service uses encryption, signature or other security protocols, select the **Security for Request** and **Security for Response** pages to configure the security for this particular service request or to open the WSDL security editor.

What to do next

After creating elements, you can use the test editor to edit service requests. You can create a service response element to test the performance and behavior of the service. You can also replace some content values with datapool variables or a references.

Updating a service response from the service

While you are developing a service test, you can send a request from the test editor to record or update the response element.

Before you begin

Service response elements are children of service request elements. Service tests use response elements to measure the response time between a call and the corresponding response. Response elements can also contain verification points. You can click **Update Response** in the request element to complete one of the following actions:

- Record a response from the service: This method sends the request and records the actual response from the service. For services that use the IBM® WebSphere® MQ or JMS transport protocols, multiple responses can be recorded.
- Update the current response content: If a response exists, its contents are replaced. If multiple responses are received, the number and order of the responses are updated.

Important: After updating the response content, data correlation or verification points that referred to replaced content might no longer work.

You can use the test editor to create or edit response elements in a service test. There are three methods of adding a service response:

- Generate a response from Web Services Description Language (WSDL): If the service uses WSDL, then the response is created with the content structure that the WSDL specifies.
- Add a text response: In this response type, you specify free formatted content for the response.
- Record a response from the service: This method sends the request and records the actual response from the service.

WebSphere MQ and JMS requests can contain multiple response elements.

Procedure

To add a response element to a service test:

1. Open the test in the test editor, and select a service request element.
2. On the Test Element Details page, click **Update Response**. Alternatively, right-click the service request element, and click **Add > Response from Request**. This action performs the service request. If the request is valid, the Update Response window opens and displays the response data.
3. In the Return Preview window, review the content of the response to ensure that it is correct. For the WebSphere MQ and JMS protocols, if multiple responses are received, then click the arrows to view each response.
 - A. Click the **Message** tab to view the contents of the response in the Form, Tree or Source view.
 - B. Click the **Attachment** tab to view any resource files that were attached to the response.

- C. Click the **Response Properties** tab to view the properties of the response.
4. To use the received response in the test, click **Update Test**. This creates the response elements as a child of the request element or updates the existing response elements with the new data.

What to do next

After creating or updating response elements, you can create verification points on the response contents to test the behavior of the service.

Related tasks:

[Manually adding a response element](#)

[Verifying application behavior](#)

Manually adding a response element

You can add service response elements to specify the received content of a service request. You can use the test editor to create or edit response elements in an existing service test.

Before you begin

Service response elements are children of service request elements. Service tests use response elements to measure the response time between a call and the corresponding response. Response elements can also contain verification points. IBM® WebSphere® MQ and JMS requests can contain multiple response elements. Depending on the type of request, you can manually create several types of response elements:

- Response from Web Services Description Language (WSDL): For web services, this response type uses the WSDL file to create the specified XML structure of the response.
- XML response: This response type creates an empty response element in which you must manually create the expected XML structure. You can use an XML Schema Definition (XSD) document from the XSD catalog to assist you.
- Text response: This response type creates an empty response element, which can contain freely formatted text.

Alternatively, you can automatically create and update response content by recording the actual response content that the service returns. See [Updating a service response from the service](#) for more information.

Procedure

To add a response element to a service test:

1. Open the test in the test editor, and select a service request element.
2. Create one of these elements:
 - For web service requests, click **Add > Response from WSDL**.
 - If the expected response contains XML content, click **Add > XML Response**.
 - If the expected response contains plain text, click **Add > Text Response**.This action creates the corresponding response element in the test. If the request uses the WebSphere MQ or JMS format, then you can create multiple responses.
3. Edit the message content of the response element to reflect to actual content that the service returns.
 - A. Click the **Message** tab to view the contents of the response in the Form, Tree or Source view.
 - B. Click the **Attachment** tab to view any resource files that were attached to the response.
 - C. Click the **Response Properties** tab to view the properties of the response.

What to do next

After creating a message return, you can create verification points on the contents to test the behavior of the service.

Related tasks:

Updating a service response from the service
Verifying application behavior

Editing WSDL security profiles

To ensure that your service test uses the correct security protocols to access a SOAP-based service, you must specify a security profile for the (Web Service Description Language) WSDL file. After a security profile is set up, it can be reused in multiple web service calls.

- **WSDL security editor overview**

With the WSDL security editor you can create the SOAP algorithm stacks that are associated with a web service operation. Algorithm stacks contain digital certificate information and the security algorithms that are applied to messages to perform secure communication with a web service.

- **Creating security profiles for WSDL files**

You can create SOAP security profiles for use with web service calls or message returns that require message encryption, signature or other advanced security algorithms.

- **Using a security policy**

The WS-Policy specification enables web services to use XML to publish their security policies either as part of the Web Services Description Language (WSDL) file (compliant with the WS-PolicyAttachment specification) or as a separate XML document. With the WSDL Security Editor, you can create a security profile that uses a policy that complies with the WS-Policy specification.

- **Implementing a custom security algorithm**

You can define your own security algorithms for SOAP security profiles by implementing custom security Java™ interfaces that can be used in the WSDL security editor. With custom security algorithms, you can implement proprietary security algorithms that transform the XML before sending and after receiving message content.

- **Adding WS-Addressing to a security configuration**

The WS-Addressing specification provides transport-neutral mechanisms that enable SOAP-based web services to communicate addressing information. You can use WSDL security algorithms to add WS-Addressing to your service tests.

WSDL security editor overview

With the WSDL security editor you can create the SOAP algorithm stacks that are associated with a web service operation. Algorithm stacks contain digital certificate information and the security algorithms that are applied to messages to perform secure communication with a web service.

After you create an algorithm stack, you associate it with an operation that is specified in the Web Services Description Language (WSDL) file of the web service. Algorithm stacks remain available in the workspace and you can reuse them with other WSDL files. You can also edit a test to make the same web service call several times with different security configurations.

You use the WSDL security editor to create and edit security configurations. The WSDL security editor contains two pages that correspond to the steps of setting up a security configuration:

- Describing a security stack
- Associating a security stack with each WSDL operation

Algorithm stacks

Algorithm stacks contain one or several algorithm blocks that are arranged in a sequence of steps. Each algorithm block modifies or transforms the message content. Algorithm blocks can add timestamps to, add tokens to, encrypt, or sign messages.

Use the Algorithm Stacks page of the WSDL security editor to create stacks for service requests and responses. When a message is sent or received, each algorithm block in the stack is executed in the specified order. For example, you can define a request stack for outgoing requests that adds a timestamp, signs, and then encrypts the message content, and you can define a response stack that decrypts incoming responses. You can create as many algorithms as your application requires.

You can edit algorithm blocks and move them up and down in the stack. Encryption and signature blocks can use keystores for digital certificates. Some algorithm blocks display messages that help you enter correct information. If the contents of the algorithm block are invalid, an error icon is displayed.

Raw transaction data view

When a stack is associated with a service request or response, viewing the results of each transformation step that is applied to the XML message content can be useful. You can use the Raw Transaction Data view to look at the message content before and after each algorithm in the stack.

Digital certificate keystores

You can add digital certificate keystores to a security stack to use with encryption or signature algorithms. Keystores must be declared with their associated passwords before the algorithms that use them. Digital certificates are contained in Java™ keystore files (KS, JKS, JCEKS, PKCS12, and PEM) that must be located in your workspace.

Associating stacks with WSDL operations

Use the Algorithms by WSDL operations page of the WSDL security editor to associate a security algorithm stack with each web service call and message return in the WSDL file.

Related tasks:

[Creating security profiles for WSDL files](#)

[Using a security policy](#)

[Implementing a custom security algorithm](#)

[Adding WS-Addressing to a security configuration](#)

Creating security profiles for WSDL files

You can create SOAP security profiles for use with web service calls or message returns that require message encryption, signature or other advanced security algorithms.

Before you begin

Before creating a security profile, you must have a Web Services Description Language (WSDL) file in your workspace.

If the security profile uses digital certificates for encrypting or signing requests or responses, you must have the corresponding keystore files (KS, JKS, JKECS, PKCS12, or PEM) in your workspace.

Procedure

1. In the test navigator or project explorer, right-click the WSDL file and select **Configure WSDL Security**. This opens the WSDL security editor.
2. Click the **Security Algorithms** tab. Security profiles are described by adding elements to a stack. When a service request is sent or a response is received, each element in the stack is applied to the message in the specified order. If necessary, create one security profile for outgoing requests and one for incoming responses.
3. In the **Security Algorithms** area, click **Add** to create a new algorithm stack, and click **Rename** to change the default name.
4. In the **Algorithm Stack Details** area, click **Add** to add a new algorithm element to the stack. You can add time stamps, user-name tokens, encryption, or signatures.
5. Edit each element in the stack according to the requirements of the web service. Encryption and signature stack elements can be applied to portions of the web service call or message return document by specifying an Xpath query in **User Xpath part selection**. For example, you can encrypt one XML element with one encryption stack element, and another element with another stack element. You can use the Web Service Protocol Data view to help identify the correct Xpath query for this option.
You can check that the security stack is valid by clicking **Tools > Validate Selected Algorithm**.
6. When all the stack elements are complete, ensure that the execution order is correct. If necessary, use the **Up** and **Down** buttons to change the order of elements in the stack.
7. Repeat steps 4 through 7 to create as many algorithms as are required for security profile.
8. Click the **Algorithms by WSDL Operations** tab. This page enables you to associate a security profile with each request or response operation in the WSDL.
9. In the **WSDL Contents** column, select a service request or response.
10. In the **Algorithm Stack** column, select a security profile from the list. If necessary, click **<<** to open the stack on the Security Algorithms page.

What to do next

After saving the security profile, the Web Service Protocol Data view displays the effect of the security profile on the XML data of the web service.

Related concepts:

[WSDL security editor overview](#)

Related tasks:

[Using a security policy](#)

[Implementing a custom security algorithm](#)

[Adding WS-Addressing to a security configuration](#)

[Using a security policy](#)

[Adding WS-Addressing to a security configuration](#)

[Implementing a custom security algorithm](#)

Related reference:

[WSDL security editor reference](#)

Using a security policy

The WS-Policy specification enables web services to use XML to publish their security policies either as part of the Web Services Description Language (WSDL) file (compliant with the WS-PolicyAttachment specification) or as a separate XML document. With the WSDL Security Editor, you can create a security profile that uses a policy that complies with the WS-Policy specification.

Before you begin

Before creating a security configuration, you must have a WSDL file in your workspace.

If the security policy uses digital certificates for encrypting or signing requests or responses, you must have the corresponding keystore files (KS, JKS, JKECS, PKCS12, or PEM) in your workspace.

When you import a WSDL that contains a policy (with WS-PolicyAttachment), a security profile is automatically generated for each operation in the WSDL security editor.

Procedure

1. In the test navigator or project explorer, right-click the WSDL file, and select **Configure WSDL Security**. This opens the WSDL security editor.
2. Click the **Security Algorithms** tab. Security profiles are described by adding elements to a stack. When a service request is sent or a response is received, each element in the stack is applied to the message in the specified order.
3. In the **Security Algorithms** area, click **Add** to create a profile, and click **Rename** to change the default name.
4. In the **Algorithm Stack Details** area, click **Add > WS-Policy** to add the WS-Policy element to the stack. You can also add time stamps, user-name tokens, encryption, or signatures.
5. If the policy is included in the WSDL file, click **Use policy included in WSDL (WS-PolicyAttachment)**, and edit the WS-Policy settings as required:
 - **Policy**
 - If you are not using the WS-PolicyAttachment specification, specify the XML policy file. Click **Browse** to add a policy file from the workspace or to import a policy file.
 - **Signature configuration**
 - Select this option to specify a keystore for any signature that is specified in the policy. Click **Edit Security** to add a keystore from the workspace or to import a keystore.
 - **Encryption configuration**
 - Select this option to specify a keystore for any encryption that is specified in the policy. Click **Edit Security** to add a keystore from the workspace or to import a keystore.
 - **Decryption configuration**
 - Select this option to specify a keystore for any decryption that is specified in the policy. Click **Edit Security** to add a keystore from the workspace or to

import a keystore.

- **Retrieve token from security token server (WS-Trust)**

- Select this option, and click **Configure** to specify a Security Token Server (STS) to use with the policy.

- **Additional properties**

- Use this table to specify settings for the advanced properties or specific implementations of the WS-Security specification. Click **Add** to add a property name and to set a value.

6. Check that the security profile is valid by clicking **Tools > Validate Selected Algorithm**.
7. Click the **Algorithms by WSDL Operations** tab. On this page, you can associate a security profile with each request or response operation in the WSDL.
8. In the **WSDL Contents** column, select a web service request or response operation.
9. In the **Algorithm Stack** column, select a security profile from the list. If necessary, click **<<** to open the stack on the Security Algorithms page.

What to do next

After saving the security profile, the Web Service Protocol Data view displays the result of the security profile on the XML data of the web service.

Related concepts:

[WSDL security editor overview](#)

Related tasks:

[Creating security profiles for WSDL files](#)

[Implementing a custom security algorithm](#)

[Adding WS-Addressing to a security configuration](#)

[Creating security profiles for WSDL files](#)

[Adding WS-Addressing to a security configuration](#)

[Implementing a custom security algorithm](#)

Implementing a custom security algorithm

You can define your own security algorithms for SOAP security profiles by implementing custom security Java™ interfaces that can be used in the WSDL security editor. With custom security algorithms, you can implement proprietary security algorithms that transform the XML before sending and after receiving message content.

Before you begin

The custom security interface and the JAR file that contains it are provided with the product in the `customsecuritydefinition` folder of the `com.ibm.rational.ttt.common.models.core` plugin. You need these interfaces to create your own algorithms. If you are using IBM® Rational® Performance Tester or IBM Rational Service Tester for SOA Quality,

Procedure

1. In the test navigator or project explorer, create a new Java class in your web service test project folder.
2. Implement a security algorithm in Java using the following interface:/**

```
* *****
* IBM Confidential
*
* (c) Copyright IBM Corporation. 2008. All Rights Reserved.
*
* The source code for this program is not published or otherwise
* divested of its trade secrets, irrespective of what has been
* deposited with the U.S. Copyright Office.
* *****
*
*/

package com.ibm.rational.test.lt.models.wscore.datamodel.security.xmlsec;

import java.util.Properties;
import org.w3c.dom.Document;

public interface ICustomSecurityAlgorithm {

/**
 * The following methods can be used in both case:
 * Execution in the workbench and execution of the test.
 */

/**
```

```

* Called to process de Document that is sent over a transport.
* @param subject
*/
void process(Document subject);
/**
* Called to un process a document that is received from a server.
* @param subject
*/
void unProcess(Document subject);

/**
* Properties defined in the UI of the CustomSecurityAlgorithm.
* @param map
*/
void setProperties(Properties map);

/**
* The following methods can only be used in terms of cast to test service interface,
* or in terms of access to the previous XML information, when the jar containing
* the custom security algorithm is deployed in the performance test project. In
* this case you cannot use the algorithm directly from the workbench.
*/

/**
* This object corresponds to the ITestExecutionService object.
* This applies only to an algorithm that must link to the execution of the test.
* If you plan to use this object you will need to deploy the jar containing the
* implementation into your performance test project and not directly into the JRE.
*
* In case of a need of the previous xml document received from the execution you can
* obtain the value using:
* IDataArea area = ((ITestExecutionService)executionObject).findDataArea(IDataArea.VIRTUALUSER);
*String previousXML = (String) area.get("PREVIOUS_XML"); //$NON-NLS-1$
*
*/
void setExecutionContext(Object executionObject);

```

The `process` method modifies the XML before it is sent to the server.

The `unprocess` method modifies the XML after it is received from the server.

The `setProperties` method retrieves any properties that are defined in the security editor for this custom security interface.

The `setExecutionContext` method is called during test with the object `ITestExecutionServices` that corresponds to the message using this custom security interface.

3. The custom security interface can be used either in the WSDL security editor for web services or in XML call elements in the **Local XML security** tab.
 - If you are testing a WSDL-based web service, right-click the WSDL file in the test navigator or project explorer to open the WSDL security editor, select the

Security Algorithms page; then, under **Details of selected security algorithm stack**, click **Add > Custom Security Algorithm**.

- If you are testing an XML call, open the XML call element in the test editor, select the **Local XML Security** tab, and then, click **Add > Custom Security Algorithm**
4. In custom security, click **Browse Class** to select the class name of the custom security algorithm, for example : `ICustomSecurityAlgorithm`.
 5. Type an **Algorithm name** for the custom security algorithm.
 6. In the properties list, use **Add**, **Remove**, or **Edit** to specify any properties that are used by the `setProperties` method in your custom security algorithm.

What to do next

After saving the security configuration or the call element, the Web Service Protocol Data view displays the effect of the security algorithm on the XML data of the web service.

Related concepts:

[WSDL security editor overview](#)

Related tasks:

[Creating security profiles for WSDL files](#)

[Using a security policy](#)

[Adding WS-Addressing to a security configuration](#)

Related reference:

[WSDL security editor reference](#)

Adding WS-Addressing to a security configuration


The WS-Addressing specification provides transport-neutral mechanisms that enable SOAP-based web services to communicate addressing information. You can use WSDL security algorithms to add WS-Addressing to your service tests.

Before you begin

Before adding WS-Addressing to a security configuration, you must have a service test with requests and responses that are related to a valid WSDL.

Procedure

To add WS-Addressing to a WSDL security algorithm:

1. Open the test, select a service request, and in the Raw Transaction Data view, select **Enable the display of the XML document after the security processing**.
2. On the Request Stack page, click **Edit WSDL Security** . **Tip:** If you need to edit separate security or processing algorithms for incoming responses, click **Show Response Stack** to add a Response Stack page to the editor.
The WSDL security editor opens.
3. Select the **Algorithm Stacks** page of the WSDL security editor, and in the **Security Algorithm** list, select or create a security algorithm.
4. In the **Stack Contents** list, click **Add > WS-Addressing** and specify the settings that are implemented by the service.
 - **WS-Addressing Algorithm**
 - Use this block if your service uses either WS-Addressing 2004/08 or the WS-Addressing 1.0 Core standard.
 - **Namespace**
 - Specify the namespace for either WS-Addressing 2004/08 or WS-Addressing 1.0 Core.
 - **Action if request uses WS-Addressing**
 - Select the action to complete if WS-Addressing is already in the request.
 - **Replace anonymous address in Reply-to with:**
 - Select this option to generate the specified address in the Reply-to header instead of an anonymous address.
 - **Remove WS-Addressing from response**
 - Select this option to strip any WS-Addressing headers from the response.
5. Save the WSDL security algorithm, and select the test editor. The WS-Addressing namespace and header XML content is displayed in the Raw Transaction Data view.

Related concepts:

[WSDL security editor overview](#)

Related tasks:

[Creating security profiles for WSDL files](#)
[Using a security policy](#)
[Implementing a custom security algorithm](#)
[Creating security profiles for WSDL files](#)
[Implementing a custom security algorithm](#)

Related reference:

[WSDL security editor reference](#)

Testing asynchronous services

- [Asynchronous service testing overview](#)

Asynchronous services use a callback interaction pattern for inter-object communications. Asynchronous services can be used, for example, in publish-subscribe systems that are provided by message-oriented middleware vendors or in system and device management domains.

- [Creating an asynchronous request structure](#)

You can create an asynchronous request based on the WS-Notification specification, which contains an callback structure.

- [Adding an asynchronous callback to a service request](#)

To test a proprietary asynchronous service that does not implement the WS-Notification specification, you can add an asynchronous callback to a service request or XML request.

Related concepts:

[Web service test editor overview](#)

Related tasks:

[Verifying application behavior](#)

[Editing WSDL security profiles](#)

[Creating a reliable messaging call structure](#)

Related information:

[Adding elements to a service test](#)

Asynchronous service testing overview

Asynchronous services use a callback interaction pattern for inter-object communications. Asynchronous services can be used, for example, in publish-subscribe systems that are provided by message-oriented middleware vendors or in system and device management domains.

WS-Notification services

Asynchronous services are standardized in the WS-Notification specifications:

- WS-BaseNotification defines the web services interfaces for NotificationProducers and NotificationConsumers. This specification includes standard message exchanges that are implemented by service providers that want to act in these roles, along with the associated operational requirements.
- WS-BrokeredNotification defines the web services interface for a NotificationBroker . A NotificationBroker is an intermediary which, among other things, enables entities that are not service providers themselves to publish messages. It includes standard message exchanges that are implemented by NotificationBroker service providers, along with the associated operational requirements of service providers and requestors that participate in brokered notifications.
- WS-Topics defines a mechanism to organize and categorize items of interest for subscription known as topics. These are used in conjunction with the notification mechanisms defined in WS-BaseNotification and WS-BrokeredNotification.

You can test web services and XML services that implement the WS-Notification specification by creating an asynchronous request inside a test. The asynchronous request contains the interfaces for the corresponding WS-Notification specification, along with a callback structure.

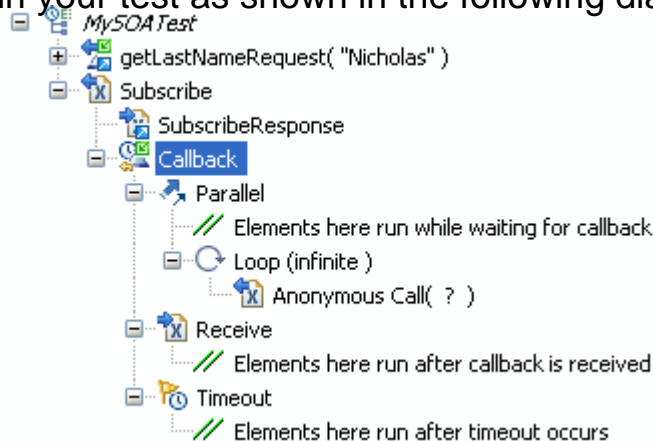
Proprietary asynchronous services

You can test proprietary asynchronous services that do not implement WS-Notification specifications. To test these services, you manually create a service request that contains the interfaces for the service, and then, you can add the asynchronous callback structure to the request.

The XML data of the asynchronous request must contain an endpoint that specifies the URL of the callback receiver. During the test, this endpoint is used to redirect the callback to the tester instead of the real receiver.

Callback structure

To test asynchronous services, you must create an asynchronous request structure in your test as shown in the following diagram:



A web service request or a plain XML request provides the subscription action and contains a callback element, which describes the behavior of the test in three states:

- Parallel contains test elements that are run after the subscription request and while waiting for the notification response.
- Receive contains test elements that are run when the notification response has been received from the service.
- Timeout contains test elements that are run if the notification response is not received after a delay that is specified in the callback element.

When everything contained in the parallel, receive, and timeout elements have finished running, the run continues with the next element in the test after the asynchronous request.

The method for generating the asynchronous callback structure in the test depends on whether the asynchronous service uses the WS-Notification specification:

- WS-Notification services: Create the asynchronous request in the test.
- Proprietary services: Manually create a web service request or XML request in the test, and then add the asynchronous callback structure to the request.

Related tasks:

[Creating an asynchronous request structure](#)

[Adding a service request](#)

[Adding an asynchronous callback to a service request](#)

Related reference:

[Service callback details](#)

[Service parallel details](#)

[Service receive details](#)

[Service timeout details](#)

Creating an asynchronous request structure

You can create an asynchronous request based on the WS-Notification specification, which contains an callback structure.

Procedure

1. In the test editor, select the test, and click **Add**, and then click **Specification-based Structure**. The New Web Service Test wizard opens.
2. On the Web Services Specification Selection page, Select **WS-Notification**, and click **Next**.
3. On the WS-Notification Details page, if the service has a Web Services Description Language (WSDL) file, click **Add** to associate it with the call.
4. Specify the **Subscription identifier**. You can select default identifiers for Websphere Application Server or Apache Muse; or if your service does not use a standard identifier, you can select **Custom**, and type the **Name** and **Namespace** of the identifier.
5. In the **Topic** area, replace the default **Name** and **Namespace** values with those of topic of your service.
6. Specify the **Subscription duration**. Because this is a test environment, the subscription expires after the specified delay to save server resources.
7. If this is a WS-BrokeredNotification service, which implements a notify call when the subscription is received, you can select **Add notify call**, and type the message to be sent.
8. Click **Next**.
9. On the Configure Protocol page, select a **Protocol configuration**, and specify the options of the configuration. Select **Generate SOAP 1.2 envelope** if you are testing a SOAP 1.2 web service.
10. Click **Finish**. This action generates in the test editor a web service call or an XML request with a callback structure that contains a parallel, a receive, and a timeout element.

What to do next

In the callback structure, add test elements to the parallel, receive, and timeout elements to specify the behavior of the test:

- Parallel contains test elements that are run after the asynchronous call has been sent.
- Receive specifies the message return of the callback and contains test elements that are run after the callback is received.
- Timeout contains test elements that are run if the callback is not received after a specified delay.

Related concepts:

[Asynchronous service testing overview](#)

Related tasks:

[Adding a service request](#)

[Adding an asynchronous callback to a service request](#)

Related reference:

[Service callback details](#)

[Service parallel details](#)

[Service receive details](#)

[Service timeout details](#)

Adding an asynchronous callback to a service request

To test a proprietary asynchronous service that does not implement the WS-Notification specification, you can add an asynchronous callback to a service request or XML request.

Before you begin

Manually create a web service call or XML call that invokes the asynchronous service. The call must contain an endpoint that specifies the URL of the callback receiver. This endpoint is used to redirect the callback to the tester.

If the service implements the WS-Notification specification, create the asynchronous call structure with the Create New WS-Notification Request and Callback wizard instead. See [Creating an asynchronous request structure](#).

Procedure

1. In the test editor, select a web service or XML request, click **Add**, and then click **Asynchronous Callback**. The Create New Asynchronous Callback wizard opens.
2. On the Select Callback Endpoint page, select the XML element of the request where the endpoint URL of the callback is located.
3. If you have a web Services Description Language (WSDL) file for the web service, click **Next**. Otherwise, skip to step 5.
4. On the Bind Message to WSDL Port page, select a port from the WSDL file. If the WSDL file for the service is not listed, click **Add** to add a WSDL file from the workspace or to import a WSDL file.
5. Click **Finish**. This generates a callback structure that contains a parallel, a receive, and a timeout element, in the test editor.

What to do next

In the callback structure, you can add test elements to the parallel, receive, and timeout elements to specify the behavior of the test:

- Parallel contains test elements that are run after the asynchronous request has been sent.
- Receive specifies the message return of the callback and contains test elements that are run after the callback is received.
- Timeout contains test elements that are run if the callback is not received after a specified delay.

Related concepts:

[Asynchronous service testing overview](#)

Related tasks:

[Creating an asynchronous request structure](#)
[Adding a service request](#)

Related reference:

[Service callback details](#)

[Service parallel details](#)

[Service receive details](#)

[Service timeout details](#)

Creating a reliable messaging call structure

You can create a test structure dedicated to testing service calls based on the WS-ReliableMessaging specification.

Before you begin

The WS-ReliableMessaging specification provides for a series of SOAP messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures. In the context of a service test, a reliable messaging call structure consists of a series of calls that conform to the specification. The structure can be created either as a sequential list of unique service calls or a loop that contains a call element and uses a datapool to identify the unique calls.

Procedure

1. In the test editor, select the test, and click **Add**, and then click **Specification-based Structure**. The New Web Service Test wizard opens.
2. On the Web Service Specification Selection page, Select **WS-ReliableMessaging**, and click **Next**.
3. Select one or several Web Services Description Language (WSDL) files in your workspace for the web service that you want to test, and click **Next**. If necessary, you can import a WSDL file into the workspace with the **Import** push button.
4. On the Configure Protocol page, select an existing HTTP transport configuration, or click **New** to create a new configuration.
 - A. Specify the **URL** of the service, the HTTP **Method**, and **Version**.
 - B. In the **Header** table, click **Add** to specify any specific headers that need to be added to the call.
 - C. In the **Cookies** table, click **Add** to specify any specific cookies that need to be used by the call.
 - D. Click **Next**.
5. On the Sequence Options page, specify how the sequence structure will be created in the test.
 - A. In **Message count**, specify the number of calls in the list or the number loop iterations.
 - B. Select **Create service call list** to generate a list of calls with the number of messages or **Create loop with datapool** to generate a loop with a datapool. The datapool defines the call number for each call in the loop.
 - C. Select **Shuffle sequence** if you want the call numbers to be generated in a random order.
6. Click **Finish**. This action generates a reliable messaging service call structure in the test.

Related concepts:

[Web service test editor overview](#)

Related tasks:

[Verifying application behavior](#)
[Editing WSDL security profiles](#)

Related information:

[Adding elements to a service test](#)
[Testing asynchronous services](#)

Simulating services with stubs

Service stubs enable you to simulate the behavior of an actual service for a wide variety testing or integration purposes.

- **Service stub overview**

Service stubs are simulations of an actual service, which can be used to functionally replace the service in a test environment. A stub server replaces the actual application server.

- **Creating a service stub**

You can use a WSDL (Web Service Description Language) specification file to generate a service stub that can simulate the behavior of the original service and uses the exact same interface.

- **Editing a service stub**

Service stubs are generated with a single default response for each operation in the WSDL specification. You can edit the service stub to change the default responses or to add conditional responses that can simulate the actual service.

- **Deploying service stubs**

You deploy and run service stubs on a stub server, which is a small application server dedicated to running service stubs. The client application, or test, addresses the stub server instead of the actual application of the original service.

- **Recording service stub activity in a log file**

With service stub logging, you can monitor the interactions between an application and the stub server. When the option is enabled, one log file is created for each deployed stub. The log files are presented as a formatted HTML report.

- **Setting log level for service stubs**

While recording a service test, you can set the level of the log details that you want to collect for debugging purposes.

Service stub overview

Service stubs are simulations of an actual service, which can be used to functionally replace the service in a test environment. A stub server replaces the actual application server.

From the point of view of the client application, the service stub looks identical to the actual service that it simulates. To use a service stub in replacement of the actual service, you must be able to replace the URL of the original service in the client application with the URL of the stub server.

Important: For version 8.7 and later, you cannot use the schedule option of IBM® Rational® Performance Tester to deploy stub servers remotely. If you have already deployed stub servers remotely, you must install IBM Rational Service Tester for SOA Quality or Rational Performance Tester on those computers and then deploy the stub servers locally.

Use case examples

There are several cases where it can be useful to deploy a stub services instead of using the actual services for your tests:

- If you are testing a local service that uses data from another remote service, you might need to inject specific content to the service under test from the remote service. You can simulate the remote service with a service stub to ensure that the local service responds properly to some specific input.
- Some commercial services charge users for each call. If you are testing such a service, you can develop and debug your test against a stub service, which is based on the WSDL of the actual service, without being charged by the commercial service.
- During integration of a large application involving multiple clients and services, some services might not yet be operational, although their WSDL specifications are available. You can simulate the missing services with service stubs, which will allow you to proceed with the integration work.

Service stub architecture

You create a service stub by providing an existing WSDL specification. The service stub is generated with the exact same ports and bindings as the original service so that it can be addressed with exactly the same interface. Each operation in the service returns a default response of the type defined by the WSDL.

You can edit the service stub in the stub editor to change the default response or to create conditional responses that simulate the actual responses of the original service.

When you have finished editing the service stub, you can deploy it on a local stub server, which runs in the workbench. The stub server simulates an actual application server and can host multiple service stubs. You control the stub server from the stub monitor view.

Finally, to use the service stub instead of the original service, you change the URL used by the client application to point to the local stub server instead of the original application server. This URL, as well as the WSDL of the service stub, is provided in

the stub monitor view.

Related tasks:

[Creating a service stub](#)

[Editing a service stub](#)

[Deploying service stubs](#)

[Recording service stub activity in a log file](#)

[Setting log level for service stubs](#)

Creating a service stub

You can use a WSDL (Web Service Description Language) specification file to generate a service stub that can simulate the behavior of the original service and uses the exact same interface.


Before you begin

Service stubs are stored in test projects. If your workspace does not contain a test project, the test creation wizard creates one, enabling you to change its name. To store a service stub in a specific project, verify that the project exists before you create the stub.

If you are using Secure Sockets Layer (SSL) authentication, ensure that you have any required key files in your workspace.

The wizard can import WSDL files from the workspace, the file system, a remote repository, or from a URL. Ensure that the WSDL files use the correct syntax for the test environment. Service stub generation might not work with some Web Services Description Language (WSDL) files.

Procedure

1. In the workbench, click **File > New > Other > Test > Test Assets > Service Test** or click the **New Service Stub**  toolbar button.
2. Select the WSDL of the service that you want to simulate. If necessary, you can import the WSDL from the file system, a URL, or a WSRR or UDDI repository.
3. Click **Next**.
4. Select a project location and a name for the new service stub. Click **Finish**.

Results

The wizard generates a working service stub that reproduces the interface of the original service as defined in the WSDL specification. Each operation is reproduced with a default response. You can edit the service stub with the stub editor to change the default response or to create conditional responses.

Related concepts:

[Service stub overview](#)

Related tasks:

[Editing a service stub](#)

[Deploying service stubs](#)

[Recording service stub activity in a log file](#)

[Setting log level for service stubs](#)

Editing a service stub

Service stubs are generated with a single default response for each operation in the WSDL specification. You can edit the service stub to change the default responses or to add conditional responses that can simulate the actual service.

Procedure

To edit the behavior of a service stub:

1. In the test navigator, double-click the stub to open the stub editor. Each operation simulated by the stub is represented by an operation element, which contains **Case** elements that describe a condition. Each case contains a response element. Case elements are similar to test verification points and use the same presentation.
2. To change the default response of an operation:
 - A. Expand the operation and the **Case : Default** element, and then select the response element. The Case : Default element describes the response of the service stub when no other case condition is met.
 - B. Edit the **Message** content to specify the XML content returned by the service stub.
3. To add a conditional response case:
 - A. Right-click the operation and select **Add > Equals Case, Contains Case, or Query Case**. These conditional case types are similar to the Equals, Contain and Query verification points in service tests.
 - Use **Equal Case** to specify a response that is returned by the stub when the entire incoming message content fully matches the specified message content.
 - Use **Contains Case** to specify a response that is returned by the service stub when a portion of the incoming message content matches the specified message content.
 - Use **Query Case** to specify a response that is returned by the service stub when an XPath query meets the specified criteria.You can add as many case elements as necessary to simulate the behavior of the original service. Use the **Up** and **Down** buttons to change the order in which the case conditions are evaluated. Only the first matching condition is executed.

The default case cannot be removed and is always the last case element in the operation.
 - B. Select the response element and edit the **Message** content to specify the XML content returned by the service stub. Use the **Form, Tree, and Source** views to change the XML content display mode.
4. Select **File > Save** or click the **Save** toolbar button.

What to do next

When you have finished editing the service stub, you can deploy the stub to a stub server.

Related concepts:

[Service stub overview](#)

Related tasks:

[Creating a service stub](#)

[Deploying service stubs](#)

[Recording service stub activity in a log file](#)

[Setting log level for service stubs](#)

Deploying service stubs

You deploy and run service stubs on a stub server, which is a small application server dedicated to running service stubs. The client application, or test, addresses the stub server instead of the actual application of the original service.

Before you begin

The local stub server runs in the workbench on the local computer. Service stubs can be accessed locally. The local stub server is automatically stopped when you close the workbench.

To use a service stub instead of the original service, you must be able to change the endpoint of the client application or service test to replace the URL of the original application with the URL of the stub server.

Procedure

1. In the stub editor, click the **Deploy** button. Alternatively, you can right-click the stub in the test navigator and select **Deploy On > Local stub server**. This opens the Stub Monitor view.
2. In the Stub Monitor view, click **▶Run**. If you make any changes to the service stub, the stub is redeployed to the stub server after saving.
3. To add more service stubs to the stub server, click **Add** and select a service stub from the workspace.
4. Copy the URL of the service stub from the Stub Monitor view and paste it into the configuration of the client application. You can also directly access the WSDL specification of the service stub, which is a copy of the original WSDL with replaced URL endpoints.

What to do next

You can validate that the service stub is responding correctly by using the generic service client to invoke a call.

Related concepts:

[Service stub overview](#)

Related tasks:

[Creating a service stub](#)

[Editing a service stub](#)

[Recording service stub activity in a log file](#)

[Setting log level for service stubs](#)

Recording service stub activity in a log file

With service stub logging, you can monitor the interactions between an application and the stub server. When the option is enabled, one log file is created for each deployed stub. The log files are presented as a formatted HTML report.

Before you begin

You must have created one or several service stubs.

Procedure

To log service stub activity:

1. Add the following virtual machine (VM) argument to the `eclipse.ini` file: -
`DSTUB_LOG_LEVEL=log_level`. Use one of the following values for the `log_level` variable:
 - 0: Disable the log.
 - 1: Log stub activity without details.
 - 2: Log stub activity including content of sent and received messages.
 - 3: Same as level 2 with HTTP headers of received messages.
 - 4: Same as level 3 with attached files.

You can also add the following optional arguments:

- `-DSTUB_LOG_KEEP_PREVIOUS=true`: This option creates a separate log file each time the service stub is redeployed. If the value is not `true` or if the option is not present, the log file is erased if the service stub is redeployed or when the stub server is stopped.
- `-DSTUB_LOG_SERIALIZE_XML=true`: This option displays the XML content (with log levels 2, 3, and 4) without formatting or indentation. If the value is not `true` or if the option is not present, the XML content is formatted and indented in the log.

The `eclipse.ini` file is located in the same directory as the `eclipse.exe` launcher binary file that is used to run the product.

For example, to enable logging with basic content, add the following line to the end of the `eclipse.ini` file: `-DSTUB_LOG_LEVEL=2`.

2. Restart the workbench, and in the Stub Monitor window, click the **Run** icon ► to restart the stub servers.
3. If the server was launched by a schedule in the performance testing application, then corresponding logs are automatically created in the workspace. If not, complete the following steps to retrieve the log files from the stub server:
 - Important:** The stub server must be running.
 - A. After running your tests, to view the service stub log files, open the Stub Monitor, and click the tab for the stub server.
 - B. Click the **Synchronize** toolbar button for the selected server. An HTML log file is created and displayed for each deployed service stub.

The stub log reports are located in a folder named `stubLogs`, which is in the same folder as the corresponding service stub.

Related concepts:

[Service stub overview](#)

Related tasks:

[Creating a service stub](#)

[Editing a service stub](#)

[Deploying service stubs](#)

[Setting log level for service stubs](#)

Setting log level for service stubs

While recording a service test, you can set the level of the log details that you want to collect for debugging purposes.


Before you begin

You must stop the stub server.

About this task

The log level that you set in this way takes precedence over the log level setting that you specify in the [eclipse.ini](#) file.

Procedure

1. In the **Stub Monitor** view, in the Service Stubs section, click the **Edit log options** icon .
2. Select one of the log level options and click **OK**.

What to do next

Start the server again for the changes to take effect.

Related concepts:

[Service stub overview](#)

Related tasks:

[Creating a service stub](#)

[Editing a service stub](#)

[Deploying service stubs](#)

[Recording service stub activity in a log file](#)

Comparing service test response contents

After running a service test with equal or contain verifications points, you can use the Raw transaction data view to compare the actual XML or text contents of the message returns with the expected results of the verification points in the test log.

Before you begin

Before comparing results you must have run a service test with equal or contain verification points.

Procedure

To compare the actual and expected results of a service message return:

1. After running the test, open the test log.
2. Make sure that the Raw Transaction Data view is open. If necessary, click **Window > Show View > Raw Transaction Data** to open the view.
3. Select an equal or contain verification point. The Raw Transaction Data view displays a comparison between the expected data from the verification point and the actual data that is received during the run.

Related concepts:

[Web service test editor overview](#)

Related tasks:

[Adding elements to a service test](#)

[Viewing message content](#)

Related reference:

[Service test details](#)

[Service test editor preferences](#)

Service test editor preferences

- **Service test details**

In the test editor, the test element is the first element in the test suite. The settings in the test element apply to the entire test.

- **Service call details**

Service call elements contain the contents of the call and the transport information for this call. The contents are made of the SOAP envelope. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

- **XML call details**

XML call elements contain the contents of the call and the transport information for this call. The contents consist of plain XML that is transmitted over an HTTP or JMS transport. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

- **Binary call details**

Binary calls are specialized service calls that can be used to send binary messages. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

- **Text call details**

Text calls are specialized calls for sending plain text messages. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

- **Service message return details**

In the test editor, message return elements are located after every service call. Message returns describe the expected content returned by the service. You can use the information in the message return element for data correlation.

- **Service verification point details**

Verification points enable you to test the behavior of a service by checking the message return of a call against criteria. You can perform checks on the contents of the XML document of the message return, the number of nodes returned by an XPath query, or the existence of a specific attachment.

- **Service callback details**

Callback elements define the web service or XML call that contains the element as an asynchronous call. The behavior of the test after invoking the asynchronous call is specified by the parallel, receive, and timeout elements that are contained in the callback element.

- **Service timeout details**

Timeout elements describe the behavior of an asynchronous service test when the callback is not received after a specified timeout period. Timeout elements are created inside callback elements.

- **Service parallel details**

Parallel elements describe the behavior of an asynchronous web service test after the asynchronous call has been made and while the tester is waiting for a callback message return. Parallel elements are created inside callback elements.

- **Service receive details**

Service receive elements specify the callback message return from an

asynchronous web service. A receive element can contain elements that describe the behavior of the test when the callback message return is received. Receive elements are created inside callback elements.

Service test details

In the test editor, the test element is the first element in the test suite. The settings in the test element apply to the entire test.

Common options

- Datapools

- This lists details about each datapool that the test uses: the name of the datapool, the columns that are used, and the location in the test where the datapool column is referenced. Click the location to navigate there.

- Add datapool

- This adds a reference to a datapool that you want a test to use. Clicking this option is the same as clicking **Add > Datapool** with the test selected.

- Remove

- This removes the selected datapool. This option is not available if the datapool is in use.

SSL configuration

Define an SSL configuration for certificate authentication between the client and the server. SSL configurations can be used by any message request in the test. If you use multiple SSL configurations in the test, you must specify the configuration in each message request.

The default SSL configuration always trusts servers, which is equivalent to no authentication.

- SSL configuration

- Select an existing SSL configuration or create one. You can use the toolbar push buttons to create a **New** SSL configuration and to **Rename** or **Delete** existing SSL configurations. You can also **Copy** and **Paste** SSL configurations to and from the SSL editor and the test editor.

- Server Authentication

- This section describes how the client trusts the server.

- Always trust server

- Select this option if no authentication is required or to ignore server certificates so that all servers are trusted. If you are using single authentication and you want to accept trusted servers only, then disable this option and specify a truststore that contains the trusted server certificates.

- Client truststore

- When you are using single authentication, the client truststore contains the certificates of all trusted servers. Click **Browse** to specify a KS, JKS, or JCEKS file containing valid certificates of the trusted servers.

- Password

- If the client truststore file is encrypted, type the password required to access the file.

- Mutual Authentication

- This section describes how the server trusts the client in addition to server authentication.

- **Use client-side certificate**

- If you are using double authentication, select this option to specify a keystore containing the client certificate. This certificate allows the server to authenticate the client.

- **Client certificate keystore**

- Click **Browse** to specify a KS, JKS, or JCEKS file containing a valid certificate that authenticates the client.

- **Password**

- If the client truststore file is encrypted, type the password required to access the file.

- **Select trust alias for Mutual Authentication**

- Select an alias to be used for the SSL configuration. There could be multiple aliases in a keystore for different security certificates. Choose an appropriate alias for a user. You can also use datapool to store aliases that you can apply to virtual users at run time.

Protocol Configuration (HTTP)

The HTTP configuration page of the test element specifies the information that your server libraries require to execute the HTTP send and receive functions.

An HTTP configuration can be used by any message call in the test. If you are using multiple protocol configurations in the test, you must specify the configuration for each message call.

- **Use HTTP Keep Alive**

- Select this option to keep the HTTP connection open after the request. This option is not available if you are using IBM® Rational® AppScan®.

- **Use SSL**

- Select this option to use an SSL configuration. Click **Configure SSL** to create an SSL configuration or select an existing configuration.

- **Platform Authentication**

- In this section, specify the type of authentication that is required to access the service. Select **None** if no authentication is required.

- **Basic HTTP authentication**

- Select this option to specify the **User Name** and **Password** that are used for basic authentication.

- **NTLM authentication**

- Select this option to use the Microsoft NT LAN Manager (NTLM) authentication protocol. NTLM uses challenge-response authentication. This view lists what is negotiated (supported by the client and requested of the server) and what is authenticated (the client reply to the challenge from the server).

- **Kerberos authentication**

- Select this option to use the Kerberos authentication protocol between the client and server.

- **Connect through proxy server**

- If the HTTP connection needs to go through a proxy server or a corporate firewall, specify the **Address** and **Port** of the proxy server. If the proxy requires authentication, select either **Basic proxy authentication** or **NTLM proxy authentication**.

- **Proxy authentication**

- In this section, specify the type of authentication that is required to access the proxy. Select **None** if no authentication is required.

- **Basic proxy authentication**

- Select this option to specify the **User Name** and **Password** that are used for basic authentication.

- **NTLM proxy authentication**

- Select this option to use the Microsoft NT LAN Manager (NTLM) authentication protocol. NTLM uses challenge-response authentication. This view lists what is negotiated (supported by the client and requested of the server) and what is authenticated (the client reply to the challenge from the server).

- **Custom class**

- Select this option if the communication protocol requires complex, low-level processing with a custom Java™ code to transform incoming or outgoing messages. Click **Browse** to select a Java class that uses the corresponding API. This option is not available in IBM Security AppScan.

Protocol Configuration (JMS)

The Java Message Service (JMS) configuration page of the test element specifies the information that your server libraries require to execute the JMS send and reception.

A JMS configuration can be used by any message call in the test. If you are using multiple protocol configurations within the test, you must specify the configuration in each message call.

- **Destination style**

- This is the style of the JMS destination. Select either **Topic** or **Queue**.

- **End-point address**

- This is the address of the destination.

- **Use temporary object**

- Select this option to send the JMS destination as a temporary object. For a JMS queue, a temporary JMS queue is sent in the message.

- **Reception point address**

- If **Use temporary object** is disabled, specify the JMS address of the destination endpoint.

- **Basic authentication**

- Select this option to specify the **User Name** and **Password** that are used for basic authentication.

- **Custom adapter class name**

- Set up a custom Java Naming and Directory Interface (JNDI) vendor adapter for this configuration. To use a custom adapter, you must write a Java class that extends the Axis class and methods. Specify the name of your custom adapter class in **Adapter class name**.

- **Text message**

- Specify whether the message is a text or a byte message.

- **Context factory properties**

- Edit the properties for a context factory. Click **Add** to add string properties to the context factory configuration.

- Connector properties

- Edit the properties for a connector. Click **Add** to add string properties to the connector configuration. The product supports the following connectors:
 - JMS priority
 - JMS delivery mode
 - JMS time to live

Protocol Configuration (WebSphere MQ)

The WebSphere® MQ configuration page of the test element specifies the information that your server libraries require to execute the WebSphere MQ transport send and receive functions.

An MQ configuration can be used by any message call in the test. If you are using multiple protocol configurations in the test, you must specify the configuration for each message call.

- Queue Manager

- Use this area to specify queue manager options for the service.
 - **Queue manager name**
 - Specify the name of the queue manager to which to send the request.
 - **Use local queue manager**
 - Select this option to use a local queue manager. If you disable this option, specify the following information:
 - **Queue manager address**
 - Specify the IP address or host name of the remote WebSphere MQ server.
 - **Queue manager port**
 - Specify the listener port of the remote WebSphere MQ server.
 - **Client channel**
 - Specify the server-connection mode channel of the remote queue manager.

- Queues

- Use this area to specify the send queue options for the service.
 - **Send queue name**
 - Specify the name of the queue that the queue manager manages.
 - **Use temporary queue for response**
 - Specifies whether the WebSphere MQ server creates a temporary queue. If selected, the temporary queue is created for the sole purpose of receiving specific messages, and then deleted.
 - **Receive queue name**
 - If **Use temporary queue** is cleared, this option specifies the queue manager that is specified on the **Queue manager name** line. The specified queue manager must manage this queue. You can specify multiple queue names by using a semicolon (;) as a separator.

- Use RFH2 header

- Select whether to use the transport for SOAP over MQ feature that is provided by WebSphere MQ. This feature uses a predetermined MQ message format (RFH2); therefore, when selected, other **Message Descriptor** options are disabled.

- **SSL connection**

- Select this option to use an SSL configuration if a **Client Channel** setting refers to a secure channel. Click **Open SSL Editor** to create an SSL configuration or **Change** to change the SSL configuration that is associated with the current test. If the WSDL that you use to create the message request uses a supported JMS URI to point to the WebSphere MQ server, then the SSL configuration is created automatically. If the test generator is unable to create the SSL configuration, you must create a new one manually.

If the WSDL is generated with the WebSphere MQ service (amqwdeployWMSservice), you must edit the WSDL to change the transport binding from HTTP to JMS to prevent the test generator from producing an HTTP configuration.

- **Cipher suite**

- Specify the cipher suite that is used in the channel configuration.

- **Message Descriptor**

- Configure the fields of the request. You can replace a subset of an MQ message descriptor with a custom format for use with other server types, specifically when using an XML message request. Refer to WebSphere MQ documentation for details about message descriptors.

- Use the **Message Properties** table to specify the following MQ message properties:

- JMSXDeliveryCount
- JMSXGroupSeq
- JMS_IBM_Report_Exception
- JMS_IBM_Report_Expiration
- JMS_IBM_Report_COA
- JMS_IBM_Report_COD
- JMS_IBM_Report_PAN
- JMS_IBM_Report_NAN
- JMS_IBM_Report_Pass_Msg_ID
- JMS_IBM_Report_Pass_Correl_ID
- JMS_IBM_Report_Discard_Msg
- JMS_IBM_MsgType
- JMS_IBM_Feedback
- JMS_IBM_PutApplType
- JMS_IBM_Encoding
- JMS_IBM_Last_Msg_In_Group

For more information about these properties, refer to the IBM WebSphere MQ documentation.

- **Target service**

- When using Microsoft .NET framework with the SOAP over MQ feature of WebSphere MQ, specify the name of the target service for the WSDL.

Related concepts:

[Web service test editor overview](#)

Related tasks:

[Adding elements to a service test](#)

[Comparing service test response contents](#)

Related reference:

[Service call details](#)

[XML call details](#)

[Binary call details](#)

[Text call details](#)

[Service message return details](#)

[Service verification point details](#)

[Service callback details](#)

[Service timeout details](#)

[Service parallel details](#)

[Service receive details](#)

[Service test editor preferences](#)

Service call details

Service call elements contain the contents of the call and the transport information for this call. The contents are made of the SOAP envelope. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

Call Settings

- Update node name automatically

- Select this option to automatically rename the request in the Test Contents view.

- Name

- Name of the request. You can use custom code or datapool to change name.

- Do not wait for response

- Select this option to skip directly to the next request in the test after the current request is sent.

- Operation and WSDL Name

- These identify the WSDL name and operation to which the service request is binded.

- WSDL Resource

- This is the name of the WSDL resource in the workbench. Click the link to edit the WSDL file. If the WSDL file is missing, click the link to bind the request to a WSDL in the workspace or to import a WSDL. You can click the **Edit WSDL Security** button to edit the security policy for the WSDL or click the **WSDL Synchronization** button to update an imported WSDL with a remote WSDL.

- Time Out (ms)

- This is the timeout value in milliseconds. If no response is received after the specified time, an error is produced.

- Think Time (ms)

- This specifies the programmatically calculated time delay that is observed for each user when this test is run with multiple virtual users. Think time is a statistical emulation of the amount of time actual users spend reading or thinking before performing an action.

- Update Response

- Click this button to invoke the request with the current settings and to use the response to create a service response element or to update the existing response element.

Message

This page shows the XML content of the request and provides access to data correlation. The same content is presented in three different ways.

- Form

- This view provides a simplified view of the message that focuses on editing the values of the XML content. Use the **Schema** menu to enable assistance with editing XML content so that the XML is valid and complies with the XSD specification. In the Form view, add the XML headers that are required for standard web service calls. On the **Header** bar, click **Add (+)** to create the default XML header structure for WS-Addressing, WS-ReliableMessaging or

WS-Coordination requests, or click **More** for other standards. You can enable or disable XML header elements and specify the correct values for each XML element. Checks are performed to ensure that the XML content is valid.

Note: To add XML headers to calls in IBM® Security AppScan®, add a **Static XML Headers** algorithm on the Request Stack tab of the request.

- Tree

- This view provides a hierarchical view of the XML structure of the message, including elements, namespaces, and the associated values. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit the XML elements and namespaces in the tree.

Use **Skip if Empty** column to select the empty XML elements that you want to skip. This column is visible only if you selected the **Display the 'Skip if Empty' column in XML tree viewer** check box in **Window > Preferences > Test > Test editor > Service test**.

Click **Filter** to hide or show namespace, attribute, or text nodes, depending on your requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable **Allow only valid modifications** if you do not have an XSD or if you want to bypass the schema.

You can right-click an XML element to convert it to an XML fragment. This enables you to perform data correlation (use datapools and create references) on the entire XML fragment instead of only on the value.

- Source

- This view displays the source XML content of the message or plain text content. To format XML content, click **Format XML text**. To wrap XML content into a single line, click **Pack XML text to single line**. Similar controls are available for JSON content. **Important:** In the Source view, do not edit the tags that start with `SoaTag`. If you delete or change these tags, any references and substitutions in the test will be broken. You cannot recreate these tags after you delete them.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments. The method for using this identifier depends on your server requirements.

- MIME or DIME

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- Use MTOM transmission mechanism

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Transport

This page covers the transport settings used to send the request. The transport protocol settings apply to a transport configuration, which can be either HTTP, Java™ Message Service (JMS), WebSphere® MQ, or Microsoft .NET. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM Security AppScan, only the HTTP transport protocol is available.

- HTTP

- Select **HTTP** to use the HTTP transport for the request. At the request level, you can update a URL or SOAP action and the reference to the global configuration of a test.
 - **Protocol configuration**
 - Click **Change** to specify a predefined transport configuration or to create a configuration. HTTP transport configurations contain proxy and authentication settings that can be reused.
 - **URL**
 - Specify the URL end point of the service request.
 - **Rest mode**
 - Use this check box to split the REST URL so that it is easy to understand the different parts of REST URL. When you use this option, the main section of URL is placed in the URL field, the resource part is placed in the **Resource** field, and the parameters are placed in the **Parameters** field. Use the **Add** button to manually add more parameters.
 - **Method and Version**
 - Specify the HTTP method and version to be used to invoke the service request.
 - **Headers**
 - Specify the names and values of any custom HTTP headers that are required by the service. Click **Add**, **Edit** or **Remove** to modify the headers list.
 - **Cookies**
 - Specify the names and values of any cookies that are required by the service. Click **Add**, **Edit** or **Remove** to modify the cookies list.

- JMS

- Select **JMS** to use the Java Messaging Service transport for the request. This page enables you to add string properties that are attached to the request for a JMS configuration. These will be sent as message properties through JMS.
 - **Protocol configuration**
 - Click **Change** to specify a predefined transport configuration or to create a configuration. JMS transport configurations contain generic end point, reception point, and adapter settings that can be reused.
 - **Properties**
 - Specify the names and values of any string properties that are required by the request for the current JMS transport configuration. These are sent as message properties through JMS. Click **Add**, **Edit** or **Remove** to modify

the properties list.

- **WebSphere MQ**

- Select **MQ** to use the IBM WebSphere MQ transport for the request. This page enables you to specify the SOAP action and override the settings for the WebSphere MQ configuration selected at the test level.

- **Protocol configuration**

- Click **Change** to specify a predefined transport configuration or to create a configuration. WebSphere MQ transport configurations contain generic queue, header, and SSL settings that can be reused.

- **SOAP Action**

- Specifies the SOAP action to be used to invoke the WebSphere MQ request.

- **Override MQ protocol configuration values**

- Select this option to configure the fields of the WebSphere MQ message. You can replace a subset of an MQ message descriptor with a custom format for use with other server types, specifically when using an XML message request.

- **Customize message header**

- Select this option to specify custom headers for the transport for the SOAP over MQ feature that is provided by WebSphere MQ. This feature uses a predetermined MQ message format (RFH2), therefore, when selected, other **Message Descriptor** options are disabled.

- **Message descriptor**

- These settings replace the message descriptor and header settings of the MQ protocol configuration. Refer to WebSphere MQ documentation for information about message descriptors.

- **Microsoft .NET**

- Select **Microsoft .NET** to use the Microsoft .NET Framework transport for requests based on Windows Communication Foundation (WCF). This page enables you to override the settings for the Microsoft .NET configuration selected at the test level.

- **Item**

- Click **Add** to specify the name and value of the WCF actions that are required by the service. This table is automatically generated when you import a Microsoft .NET WSDL file. Refer to the Microsoft .NET WCF documentation for more information.

Related reference:

[Service test details](#)

[XML call details](#)

[Binary call details](#)

[Text call details](#)

[Service message return details](#)

[Service verification point details](#)

[Service callback details](#)

[Service timeout details](#)

Service parallel details
Service receive details

XML call details

XML call elements contain the contents of the call and the transport information for this call. The contents consist of plain XML that is transmitted over an HTTP or JMS transport. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

- Update node name automatically

- When enabled, this option updates the name of the XML call element in the test contents.

- One way

- This option specifies that no response from the server is expected after the call. This disables the **Update Return** button.

- Time Out (ms)

- This is the timeout value in milliseconds. If no response is received after the specified time, an error is produced.

- Think Time (ms)

- This specifies the programmatically-calculated time delay that is observed for each user when this test is run with multiple virtual users. Think time is a statistical emulation of the amount of time actual users spend reading or thinking before performing an action.

- Update Return

- This opens the Return Preview window. From this window, you can invoke the call from the workbench to create or update the message return that is associated with the call.

Message

These pages present the XML contents of the call and provide access to data correlation in three different forms

- Form

- This view provides a simplified view of the message that focuses on editing the values of the XML content. Use the **Schema** menu to enable assistance with editing XML content so that the XML is valid and complies with the XSD specification. In the Form view, add the XML headers that are required for standard web service calls. On the **Header** bar, click **Add (+)** to create the default XML header structure for WS-Addressing, WS-ReliableMessaging or WS-Coordination requests, or click **More** for other standards. You can enable or disable XML header elements and specify the correct values for each XML element. Checks are performed to ensure that the XML content is valid.

Note: To add XML headers to calls in IBM® Security AppScan®, add a **Static XML Headers** algorithm on the Request Stack tab of the request.

- Tree

- This view provides a hierarchical view of the XML structure of the message, including elements, namespaces, and the associated values. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit the XML elements and namespaces in the tree.

Use **Skip if Empty** column to select the empty XML elements that you want to skip. This column is visible only if you selected the **Display the 'Skip if Empty'**

column in XML tree viewer check box in **Window > Preferences > Test > Test editor > Service test**.

Click **Filter** to hide or show namespace, attribute, or text nodes, depending on your requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable **Allow only valid modifications** if you do not have an XSD or if you want to bypass the schema.

You can right-click an XML element to convert it to an XML fragment. This enables you to perform data correlation (use datapools and create references) on the entire XML fragment instead of only on the value.

- Source

- This view displays the source XML content of the message or plain text content. To format XML content, click **Format XML text**. To wrap XML content into a single line, click **Pack XML text to single line**. Similar controls are available for JSON content. **Important:** In the Source view, do not edit the tags that start with `SoaTag`. If you delete or change these tags, any references and substitutions in the test will be broken. You cannot recreate these tags after you delete them.

Attachments

This page lists the MIME attachments that are attached to the call. The contents of this view correspond to the specification of Multipurpose Internet Mail Extensions (MIME). You can use this page to add workbench resources as MIME attachments and change properties.

The **Content ID** is the identifier that the call uses to refer to the attachments. The method for using this identifier depends on your server requirements.

Protocol

This page covers the protocol that is used to send the call. The protocol can be either HTTP or Java™ Message Service (JMS) on a message-by-message basis.

- HTTP

- This page enables you to override the HTTP settings that are attached to the call for a local HTTP configuration.

- Method

- This option enables you to specify the HTTP method of the XML call, among the following list of methods:
 - POST
 - GET
 - PUT
 - DELETE

- Version

- This option enables you to specify either HTTP 1.0 or HTTP 1.1.

- URL

- This field enables you to specify the URL of the XML call.

- Headers

- This section enables you to add headers to your call. Headers must be compatible with the specified HTTP method.
The application manages the following headers and they cannot be added:
 - User Agent
 - Host Connection
 - Cache-Control
 - Pragma
 - Content-Type
 - Content-Length

- **Cookies**

- This section enables you to manage cookies. You can add, edit and remove cookies, and create references.

- **JMS**

- This page enables you to add string properties that are attached to the call for a local JMS configuration. These will be sent as message properties through JMS.

- **MQ**

- This page enables you to override settings that are attached to the call for a local WebSphere® MQ configuration.

- **Name**

- This is the name that is displayed in the message call as a link to this protocol configuration.

- **Queue manager name**

- Specifies the name of the queue manager to which you want to send the call.

- **Queue name**

- Specifies the name of the queue that the queue manager manages.

- **Use local queue manager**

- Specifies whether the WebSphere MQ server is running on the local computer. If the server is located on a remote computer, clear this option to specify the remote MQ server details.

- **Queue manager address**

- Specifies the IP address or hostname of the remote MQ server.

- **Queue manager port**

- Specifies the listener port of the remote MQ server.

- **Client channel**

- Specifies the server-connection mode channel of the remote queue manager.

- **Use temporary queue**

- Specifies whether the MQ server creates a temporary queue. If selected, the temporary queue is created for the sole purpose of receiving specific messages, and then deleted.

- **Queue name**

- If **Use temporary queue** is cleared, this option specifies the name of the queue where message returns from the MQ server are received. The queue manager that is specified in **Queue manager name** must manage this queue. The calls and message returns are associated by the

correlation ID in the MQ message, which means that the report setting of the message is set to MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID. The server must follow this constraint. This supports the transport for SOAP feature provided by WebSphere MQ.

- Target service

- This option is for using Microsoft .NET with the IBM WebSphere MQ transport for SOAP feature. This specifies the name of the ASPX file within the .NET listener directory.

- Use RFH2 Header

- Specifies whether the SOAP messages uses an RFH2 header, which uses a predetermined MQ message format. When selected, other **Message Descriptor** options are disabled. Use this option for the WebSphere MQ transport for SOAP feature. If you are using WebSphere Integration Developer (WID) MQ binding, the binding understands messages with or without the RFH2 header.

- Message Descriptor

- This section enables you to configure the fields of the message call. You can replace a subset of an MQ message descriptor with a custom format for use with other server types, specifically when using an XML message call. See WebSphere MQ documentation for details about message descriptors.

- Use temporary queue

- This section enables you to specify a user name and password for basic authentication on the application server.

- SSL connection

- Select this option to use an SSL configuration if a **Client Channel** setting refers to a secure channel. Click **Open SSL Editor** to create a new SSL configuration, or **Change** to change the SSL configuration that is associated with the current test. If the Web Services Description Language (WSDL) that is used to create the message call uses a supported JMS URI to point to the WebSphere MQ server, then the SSL configuration is created automatically. If the test generator was unable to create the SSL configuration, you must create a new one manually.

If the WSDL was generated with the WebSphere MQ service (amqwdeployWMSservice), edit the WSDL to change the transport binding from HTTP to JMS in order to prevent the test generator to produce an HTTP configuration.

- Cipher suite

- Specify the cipher suite that is used in the channel configuration.

Local XML Security

This page allows you to add a custom security algorithm that is implemented in a Java class. Custom algorithms can be applied to the XML contents that are sent to and received from the server.

- Add, Insert, Remove, Up, and Down

- These buttons allow you to create a stack of security algorithms. Each algorithm is applied to the stack sequentially. Click **Add** to add a custom security algorithm.

- Tools

- This button allows you to change the way the algorithm stack is displayed.

- Custom Security Algorithm

- After adding a custom security algorithm to the stack. With this window, you can specify the Java class that implements the algorithm. The Java class uses the following interface:

```
/**
 * *****
 * IBM Confidential
 *
 * (c) Copyright IBM Corporation. 2008. All Rights Reserved.
 *
 * The source code for this program is not published or otherwise
 * divested of its trade secrets, irrespective of what has been
 * deposited with the U.S. Copyright Office.
 * *****
 *
 */

package com.ibm.rational.test.lt.models.wscore.datamodel.security.xmlsec;

import java.util.Properties;
import org.w3c.dom.Document;

public interface ICustomSecurityAlgorithm {

    /**
     * The following methods can be used in both case:
     * Execution in the workbench and execution of the test.
     */

    /**
     * Called to process de Document that is sent over a transport.
     * @param subject
     */
    void process(Document subject);

    /**
     * Called to un process a document that is received from a server.
     * @param subject
     */
    void unProcess(Document subject);

    /**
     * Properties defined in the UI of the CustomSecurityAlgorithm.
     * @param map
     */
}
```

```

void setProperties(Properties map);

/**
 * The following methods can only be used in terms of cast to test service interface,
 * or in terms of access to the previous XML information, when the jar containing
 * the custom security algorithm is deployed in the performance test project. In
 * this case you cannot use the algorithm directly from the workbench.
 */

/**
 * This object corresponds to the ITestExecutionService object.
 * This applies only to an algorithm that must link to the execution of the test.
 * If you plan to use this object you will need to deploy the jar containing the
 * implementation into your performance test project and not directly into the JRE.
 *
 * In case of a need of the previous xml document received from the execution you can
 * obtain the value using:
 * IDataArea area = ((ITestExecutionService)executionObject).findDataArea(IDataArea.VIRTUALUSER);
 *String previousXML = (String) area.get("PREVIOUS_XML"); //$NON-NLS-1$
 */

void setExecutionContext(Object executionObject);

```

- The `process` method modifies the XML before it is sent to the server.
- The `unprocess` method modifies the XML after it is received from the server.
- The `setProperties` method retrieves any properties that are defined in the security editor for this custom security interface.
- The `setExecutionContext` method is called during test with the object `ITestExecutionServices` that corresponds to the message using this custom security interface.
- **Custom Security Algorithm Class Name**
 - This specifies the class that implements the security algorithm. Click **Browse Class** to select a class from the workspace.
- **Algorithm Name**
 - This specifies a name for the current algorithm.
- **Properties**
 - This list specifies properties that the `setProperties` method uses in the algorithm. Use **Add**, **Remove**, or **Edit** to create the properties list.

Related reference:

[Service test details](#)

[Service call details](#)

[Binary call details](#)

[Text call details](#)

[Service message return details](#)

[Service verification point details](#)

[Service callback details](#)

Service timeout details
Service parallel details
Service receive details

Binary call details

Binary calls are specialized service calls that can be used to send binary messages. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

Message

- Update node name automatically

- Select this option to automatically rename the request in the Test Contents view.

- Do not wait for response

- Select this option to skip directly to the next request in the test after the current request is sent.

- Time Out (ms)

- This is the timeout value in milliseconds. If no response is received after the specified time, an error is produced.

- Think Time (ms)

- This specifies the programmatically calculated time delay that is observed for each user when this test is run with multiple virtual users. Think time is a statistical emulation of the amount of time actual users spend reading or thinking before performing an action.

- Update Response

- Click this button to invoke the request with the current settings and to use the response to create a binary response element or to update the existing response element.

- Source

- This page presents the binary contents of the request and provides access to data correlation. The same contents are presented in Binary and Raw ASCII views.

- Attachments

- This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties.

- Transport

- This page covers the transport protocol used to send the request. The transport protocol can be either HTTP, Java™ Message Service (JMS), or WebSphere® MQ. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM® Security AppScan®, only the HTTP transport protocol is available.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments.

The method for using this identifier depends on your server requirements.

- MIME or DIME

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- Use MTOM transmission mechanism

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Transport

This page covers the transport settings used to send the request. The transport protocol settings apply to a transport configuration, which can be either HTTP, Java Message Service (JMS), WebSphere MQ, or Microsoft .NET. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM Security AppScan, only the HTTP transport protocol is available.

- HTTP

- Select **HTTP** to use the HTTP transport for the request. At the request level, you can update a URL or SOAP action and the reference to the global configuration of a test.

- Protocol configuration

- Click **Change** to specify a predefined transport configuration or to create a configuration. HTTP transport configurations contain proxy and authentication settings that can be reused.

- URL

- Specify the URL end point of the service request.

- Rest mode

- Use this check box to split the REST URL so that it is easy to understand the different parts of REST URL. When you use this option, the main section of URL is placed in the URL field, the resource part is placed in the **Resource** field, and the parameters are placed in the **Parameters** field. Use the **Add** button to manually add more parameters.

- Method and Version

- Specify the HTTP method and version to be used to invoke the service request.

- Headers

- Specify the names and values of any custom HTTP headers that are required by the service. Click **Add**, **Edit** or **Remove** to modify the headers list.

- Cookies

- Specify the names and values of any cookies that are required by the service. Click **Add**, **Edit** or **Remove** to modify the cookies list.

- JMS

- Select **JMS** to use the Java Messaging Service transport for the request. This page enables you to add string properties that are attached to the request for a

JMS configuration. These will be sent as message properties through JMS.

- Protocol configuration

- Click **Change** to specify a predefined transport configuration or to create a configuration. JMS transport configurations contain generic end point, reception point, and adapter settings that can be reused.

- Properties

- Specify the names and values of any string properties that are required by the request for the current JMS transport configuration. These are sent as message properties through JMS. Click **Add**, **Edit** or **Remove** to modify the properties list.

- WebSphere MQ

- Select **MQ** to use the IBM WebSphere MQ transport for the request. This page enables you to specify the SOAP action and override the settings for the WebSphere MQ configuration selected at the test level.

- Protocol configuration

- Click **Change** to specify a predefined transport configuration or to create a configuration. WebSphere MQ transport configurations contain generic queue, header, and SSL settings that can be reused.

- SOAP Action

- Specifies the SOAP action to be used to invoke the WebSphere MQ request.

- Override MQ protocol configuration values

- Select this option to configure the fields of the WebSphere MQ message. You can replace a subset of an MQ message descriptor with a custom format for use with other server types, specifically when using an XML message request.

- Customize message header

- Select this option to specify custom headers for the transport for the SOAP over MQ feature that is provided by WebSphere MQ. This feature uses a predetermined MQ message format (RFH2), therefore, when selected, other **Message Descriptor** options are disabled.

- Message descriptor

- These settings replace the message descriptor and header settings of the MQ protocol configuration. Refer to WebSphere MQ documentation for information about message descriptors.

- Microsoft .NET

- Select **Microsoft .NET** to use the Microsoft .NET Framework transport for requests based on Windows Communication Foundation (WCF). This page enables you to override the settings for the Microsoft .NET configuration selected at the test level.

- Item

- Click **Add** to specify the name and value of the WCF actions that are required by the service. This table is automatically generated when you import a Microsoft .NET WSDL file. Refer to the Microsoft .NET WCF documentation for more information.

Related reference:

[Service test details](#)

[Service call details](#)

[XML call details](#)

[Text call details](#)

[Service message return details](#)

[Service verification point details](#)

[Service callback details](#)

[Service timeout details](#)

[Service parallel details](#)

[Service receive details](#)

Text call details

Text calls are specialized calls for sending plain text messages. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

Message

- Update node name automatically

- Select this option to automatically rename the request in the Test Contents view.

- Do not wait for response

- Select this option to skip directly to the next request in the test after the current request is sent.

- Time Out (ms)

- This is the timeout value in milliseconds. If no response is received after the specified time, an error is produced.

- Think Time (ms)

- This specifies the programmatically calculated time delay that is observed for each user when this test is run with multiple virtual users. Think time is a statistical emulation of the amount of time actual users spend reading or thinking before performing an action.

- Update Response

- Click this button to invoke the request with the current settings and to use the response to create a service response element or to update the existing response element.

- Source

- This page presents the plain text contents of the request and provides access to data correlation.

- Attachments

- This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties.

- Transport

- This page covers the transport protocol used to send the request. The transport protocol can be either HTTP, Java™ Message Service (JMS), WebSphere® MQ, or Microsoft .NET. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM® Security AppScan®, only the HTTP transport protocol is available.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments. The method for using this identifier depends on your server requirements.

- **MIME or DIME**

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- **Use MTOM transmission mechanism**

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Transport

This page covers the transport settings used to send the request. The transport protocol settings apply to a transport configuration, which can be either HTTP, Java Message Service (JMS), WebSphere MQ, or Microsoft .NET. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM Security AppScan, only the HTTP transport protocol is available.

- **HTTP**

- Select **HTTP** to use the HTTP transport for the request. At the request level, you can update a URL or SOAP action and the reference to the global configuration of a test.
 - **Protocol configuration**
 - Click **Change** to specify a predefined transport configuration or to create a configuration. HTTP transport configurations contain proxy and authentication settings that can be reused.
 - **URL**
 - Specify the URL end point of the service request.
 - **Rest mode**
 - Use this check box to split the REST URL so that it is easy to understand the different parts of REST URL. When you use this option, the main section of URL is placed in the URL field, the resource part is placed in the **Resource** field, and the parameters are placed in the **Parameters** field. Use the **Add** button to manually add more parameters.
 - **Method and Version**
 - Specify the HTTP method and version to be used to invoke the service request.
 - **Headers**
 - Specify the names and values of any custom HTTP headers that are required by the service. Click **Add**, **Edit** or **Remove** to modify the headers list.
 - **Cookies**
 - Specify the names and values of any cookies that are required by the service. Click **Add**, **Edit** or **Remove** to modify the cookies list.
- **JMS**
 - Select **JMS** to use the Java Messaging Service transport for the request. This page enables you to add string properties that are attached to the request for a JMS configuration. These will be sent as message properties through JMS.

- **Protocol configuration**
 - Click **Change** to specify a predefined transport configuration or to create a configuration. JMS transport configurations contain generic end point, reception point, and adapter settings that can be reused.
- **Properties**
 - Specify the names and values of any string properties that are required by the request for the current JMS transport configuration. These are sent as message properties through JMS. Click **Add**, **Edit** or **Remove** to modify the properties list.
- **WebSphere MQ**
 - Select **MQ** to use the IBM WebSphere MQ transport for the request. This page enables you to specify the SOAP action and override the settings for the WebSphere MQ configuration selected at the test level.
 - **Protocol configuration**
 - Click **Change** to specify a predefined transport configuration or to create a configuration. WebSphere MQ transport configurations contain generic queue, header, and SSL settings that can be reused.
 - **SOAP Action**
 - Specifies the SOAP action to be used to invoke the WebSphere MQ request.
 - **Override MQ protocol configuration values**
 - Select this option to configure the fields of the WebSphere MQ message. You can replace a subset of an MQ message descriptor with a custom format for use with other server types, specifically when using an XML message request.
 - **Customize message header**
 - Select this option to specify custom headers for the transport for the SOAP over MQ feature that is provided by WebSphere MQ. This feature uses a predetermined MQ message format (RFH2), therefore, when selected, other **Message Descriptor** options are disabled.
 - **Message descriptor**
 - These settings replace the message descriptor and header settings of the MQ protocol configuration. Refer to WebSphere MQ documentation for information about message descriptors.
- **Microsoft .NET**
 - Select **Microsoft .NET** to use the Microsoft .NET Framework transport for requests based on Windows Communication Foundation (WCF). This page enables you to override the settings for the Microsoft .NET configuration selected at the test level.
 - **Item**
 - Click **Add** to specify the name and value of the WCF actions that are required by the service. This table is automatically generated when you import a Microsoft .NET WSDL file. Refer to the Microsoft .NET WCF documentation for more information.

Related reference:

Service test details
Service call details
XML call details
Binary call details
Service message return details
Service verification point details
Service callback details
Service timeout details
Service parallel details
Service receive details

Service message return details

In the test editor, message return elements are located after every service call. Message returns describe the expected content returned by the service. You can use the information in the message return element for data correlation.

You can automatically generate or update the contents of a message return by clicking **Update Return** in the call element.

Message

This page shows the XML content of the request and provides access to data correlation. The same content is presented in three different ways.

- Form

- This view provides a simplified view of the message that focuses on editing the values of the XML content. Use the **Schema** menu to enable assistance with editing XML content so that the XML is valid and complies with the XSD specification. In the Form view, add the XML headers that are required for standard web service calls. On the **Header** bar, click **Add (+)** to create the default XML header structure for WS-Addressing, WS-ReliableMessaging or WS-Coordination requests, or click **More** for other standards. You can enable or disable XML header elements and specify the correct values for each XML element. Checks are performed to ensure that the XML content is valid.

Note: To add XML headers to calls in IBM® Security AppScan®, add a **Static XML Headers** algorithm on the Request Stack tab of the request.

- Tree

- This view provides a hierarchical view of the XML structure of the message, including elements, namespaces, and the associated values. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit the XML elements and namespaces in the tree.

Use **Skip if Empty** column to select the empty XML elements that you want to skip. This column is visible only if you selected the **Display the 'Skip if Empty' column in XML tree viewer** check box in **Window > Preferences > Test > Test editor > Service test**.

Click **Filter** to hide or show namespace, attribute, or text nodes, depending on your requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable **Allow only valid modifications** if you do not have an XSD or if you want to bypass the schema.

You can right-click an XML element to convert it to an XML fragment. This enables you to perform data correlation (use datapools and create references) on the entire XML fragment instead of only on the value.

- Source

- This view displays the source XML content of the message or plain text content. To format XML content, click **Format XML text**. To wrap XML content into a single line, click **Pack XML text to single line**. Similar controls are available for JSON content. **Important:** In the Source view, do not edit the tags that start with SoaTag. If you delete or change these tags, any references and substitutions in

the test will be broken. You cannot recreate these tags after you delete them.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments. The method for using this identifier depends on your server requirements.

- MIME or DIME

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- Use MTOM transmission mechanism

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Response Properties

This page lists the names and values of properties of the response.

Related reference:

[Service test details](#)

[Service call details](#)

[XML call details](#)

[Binary call details](#)

[Text call details](#)

[Service verification point details](#)

[Service callback details](#)

[Service timeout details](#)

[Service parallel details](#)

[Service receive details](#)

Service verification point details

Verification points enable you to test the behavior of a service by checking the message return of a call against criteria. You can perform checks on the contents of the XML document of the message return, the number of nodes returned by an XPath query, or the existence of a specific attachment.

Contain and equal verification points

Contain verification points return a Pass status when the message return object contains the specified XML message. *Equal* verification points return a Pass status when the message return object matches the specified XML message.

The verification occurs if the message return object is a valid XML message. The verification is performed on both the name of the XML element and final return value of the element. Attributes are not checked.

Use the **Form**, **Tree** and **Source** views to edit the message content.

- Test using XML namespace

- Select this option to perform the verification on a qualified structure, including the XML namespace, instead of the simple name. For example, if the expected XML data is:

```
<ns1:responseElement xmlns:ns1="http://www.ibm.com/wbse"></ns1:responseElement>
```

When **Namespace aware** is enabled, the verification is made on the full name of the return value:

```
<ns1:responseElement xmlns:ns1="http://www.ibm.com/wbse"></ns1:responseElement>
```

When **Namespace aware** is disabled, the verification ignores the namespace tagging and checks only the simple name of the element and the final return value:

```
<ns1:responseElement xmlns:ns1="http://www.ibm.com/wbse"></ns1:responseElement>
```

In this case, you can simplify the value of the expected XML data to:

```
<responseElement>  
></responseElement>
```

- Test XML text nodes

- Select this option to include XML text values in the verification.

- Test XML attributes

- Select this option to include XML attributes in the verification.

- Form

- This view provides an simple view of the elements of the call with their values. Use this view to quickly edit values in the form.

- Tree

- This view provides a hierarchical view of the elements of the call with their values, attributes, and the associated namespaces. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit this list.

Click the **namespace**, **attribute**, or **text** filter buttons, depending on your requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable smart editing if you do not have an XSD or if you want to bypass the schema.

You can specify standard Java™ regular expressions. In the **Regex** column,

select the line of an attribute or text value and type the regular expression in the **Value** column. For example, the following regular expression checks for a correctly formatted email address: `/^([a-zA-Z0-9_\.\\-])+\@(([a-zA-Z0-9\\-])+\.)+([a-zA-Z0-9]{2,4})+$/`

- **Source**

- This view displays the source XML document of the call. **Important:** The ID tags that are shown in the Source page refer to an internal representation for the test. If you remove these tags, you will remove any existing references and substitutions. You cannot re-create these tags after you delete them.

Query verification points

Query verification points return a Pass status when the number of nodes returned by an XML Path language query matches the expected number of nodes specified in the verification point.

The verification occurs if the message return object is a valid XML document.

- **XPath expression**

- Specify a query using the XML path language. Refer to the XPath specification for details on expressing an XPath query: <http://www.w3.org/TR/xpath>. Click **Build Expression** to open the XPath Expression Builder window.

Note: Because XPath expressions require that the qualified name has a prefix, XPath expressions will return null for the default namespace declared with `xmlns`.

- **Operator and Expected Count**

- These specify the expected number of nodes returned by the query.

- **Evaluate**

- Click this button to calculate the number of nodes based on the current input. This value automatically replaces the current **Expected count**.

Attachment verification points

Attachment verification points return a Pass status when the message return attachment matches all of the criteria specified in the verification point.

The verification occurs only if the message return object is a valid XML document.

- **Index of the attachment to be verified**

- In the case of multiple attachments, this number specifies which attachment to check.

- **Attachment size**

- This specifies the expected size of the attachment.

- **MIME type**

- This specifies the expected MIME type of the attachment.

- **Encoding**

- This specifies the expected encoding of the attachment.

XSD verification points

XSD verification points check that the content returned by the service is validated by the specified XML Schema Definition (XSD) files or Web Service Definition Language (WSDL) files that contain XSDs.

The verification occurs only if the message return object is a valid XML document.

- Add XSD

- Add an XSD to the list of validation checks.

- Add WSDL

- Add a WSDL that contains an XSD to the list of validation checks.

- Open

- Open a selected XSD or WSDL file.

Related reference:

[Service test details](#)

[Service call details](#)

[XML call details](#)

[Binary call details](#)

[Text call details](#)

[Service message return details](#)

[Service callback details](#)

[Service timeout details](#)

[Service parallel details](#)

[Service receive details](#)

Service callback details

Callback elements define the web service or XML call that contains the element as an asynchronous call. The behavior of the test after invoking the asynchronous call is specified by the parallel, receive, and timeout elements that are contained in the callback element.

- **Callback endpoint location**

- This element specifies the XML element in the asynchronous call that defines the endpoint URL of the callback receiver. During a test, this endpoint is used to redirect the callback to the tester instead of the real receiver.

- **Display full path**

- Select this option to display the extended path of the endpoint XML element.

Related concepts:

[Asynchronous service testing overview](#)

Related tasks:

[Creating an asynchronous request structure](#)

[Adding a service request](#)

[Adding an asynchronous callback to a service request](#)

Related reference:

[Service test details](#)

[Service call details](#)

[XML call details](#)

[Binary call details](#)

[Text call details](#)

[Service message return details](#)

[Service verification point details](#)

[Service timeout details](#)

[Service parallel details](#)

[Service receive details](#)

Service timeout details

Timeout elements describe the behavior of an asynchronous service test when the callback is not received after a specified timeout period. Timeout elements are created inside callback elements.

- **Timeout**

- This value specifies the timeout delay after which the test runs the elements that the timeout element contains.

- **Enable verification point for timeout**

- With this option, you can enable a verification point on the timeout. If the timeout delay is reached, the verification point reports a *fail* status in the test log.

The list displays the test elements that the timeout element contains. These are the same contents as displayed in the Test Contents of the test editor.

Related concepts:

[Asynchronous service testing overview](#)

Related tasks:

[Creating an asynchronous request structure](#)

[Adding a service request](#)

[Adding an asynchronous callback to a service request](#)

Related reference:

[Service test details](#)

[Service call details](#)

[XML call details](#)

[Binary call details](#)

[Text call details](#)

[Service message return details](#)

[Service verification point details](#)

[Service callback details](#)

[Service parallel details](#)

[Service receive details](#)

Service parallel details

Parallel elements describe the behavior of an asynchronous web service test after the asynchronous call has been made and while the tester is waiting for a callback message return. Parallel elements are created inside callback elements.

The list displays the test elements that the parallel element contains. These are the same contents as displayed in the Test Contents area of the test editor.

Related concepts:

[Asynchronous service testing overview](#)

Related tasks:

[Creating an asynchronous request structure](#)

[Adding a service request](#)

[Adding an asynchronous callback to a service request](#)

Related reference:

[Service test details](#)

[Service call details](#)

[XML call details](#)

[Binary call details](#)

[Text call details](#)

[Service message return details](#)

[Service verification point details](#)

[Service callback details](#)

[Service timeout details](#)

[Service receive details](#)

Service receive details

Service receive elements specify the callback message return from an asynchronous web service. A receive element can contain elements that describe the behavior of the test when the callback message return is received. Receive elements are created inside callback elements.

The contents of a receive element are the same as a typical message return element.

Message

This page shows the XML content of the request and provides access to data correlation. The same content is presented in three different ways.

- Form

- This view provides a simplified view of the message that focuses on editing the values of the XML content. Use the **Schema** menu to enable assistance with editing XML content so that the XML is valid and complies with the XSD specification. In the Form view, add the XML headers that are required for standard web service calls. On the **Header** bar, click **Add (+)** to create the default XML header structure for WS-Addressing, WS-ReliableMessaging or WS-Coordination requests, or click **More** for other standards. You can enable or disable XML header elements and specify the correct values for each XML element. Checks are performed to ensure that the XML content is valid.

Note: To add XML headers to calls in IBM® Security AppScan®, add a **Static XML Headers** algorithm on the Request Stack tab of the request.

- Tree

- This view provides a hierarchical view of the XML structure of the message, including elements, namespaces, and the associated values. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit the XML elements and namespaces in the tree.

Use **Skip if Empty** column to select the empty XML elements that you want to skip. This column is visible only if you selected the **Display the 'Skip if Empty' column in XML tree viewer** check box in **Window > Preferences > Test > Test editor > Service test**.

Click **Filter** to hide or show namespace, attribute, or text nodes, depending on your requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable **Allow only valid modifications** if you do not have an XSD or if you want to bypass the schema.

You can right-click an XML element to convert it to an XML fragment. This enables you to perform data correlation (use datapools and create references) on the entire XML fragment instead of only on the value.

- Source

- This view displays the source XML content of the message or plain text content. To format XML content, click **Format XML text**. To wrap XML content into a single line, click **Pack XML text to single line**. Similar controls are available for

JSON content. **Important:** In the Source view, do not edit the tags that start with `SoaTag`. If you delete or change these tags, any references and substitutions in the test will be broken. You cannot recreate these tags after you delete them.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments. The method for using this identifier depends on your server requirements.

- MIME or DIME

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- Use MTOM transmission mechanism

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Response Properties

This page lists the names and values of properties of the response.

Related concepts:

[Asynchronous service testing overview](#)

Related tasks:

[Creating an asynchronous request structure](#)

[Adding a service request](#)

[Adding an asynchronous callback to a service request](#)

Related reference:

[Service test details](#)

[Service call details](#)

[XML call details](#)

[Binary call details](#)

[Text call details](#)

[Service message return details](#)

[Service verification point details](#)

[Service callback details](#)

[Service timeout details](#)

[Service parallel details](#)

Service stub editor reference

Service stub elements are part of the service stub and can be edited in the stub editor.

- **Stub operation details**

Stub operation elements describe the format of the call that the service stub expects to receive. There is one stub operation for each operation that was detected in the WSDL specification. Each stub operation contains at least one default case element, or several case elements describing the response of the service stub depending on the incoming calls. The information from the stub operation element can be used for data correlation.

- **Stub case details**

Stub case elements enable you to specify the response of a service stub according to the content of an incoming call. You can perform checks on the contents of the XML document of the message return, the number of nodes returned by an XPath query, or the existence of a specific attachment. Each case element has an associated response element. There can be multiple case elements in a stub operation, but the Case : Default element is mandatory.

- **Stub response details**

In the stub editor, there is one response elements associated with each case element. Stub responses describe the content that is returned by the stub service, simulating the response of the original service.

Stub operation details

Stub operation elements describe the format of the call that the service stub expects to receive. There is one stub operation for each operation that was detected in the WSDL specification. Each stub operation contains at least one default case element, or several case elements describing the response of the service stub depending on the incoming calls. The information from the stub operation element can be used for data correlation.

This page presents the XML or text contents of the call and provides access to data correlation. The same contents are presented in **Form**, **Tree** or **Source** view.

- Form

- This view provides a simplified view of the message that focuses on editing the values of the XML content. Use the **Schema** menu to enable assistance with editing XML content so that the XML is valid and complies with the XSD specification. In the Form view, add the XML headers that are required for standard web service calls. On the **Header** bar, click **Add (+)** to create the default XML header structure for WS-Addressing, WS-ReliableMessaging or WS-Coordination requests, or click **More** for other standards. You can enable or disable XML header elements and specify the correct values for each XML element. Checks are performed to ensure that the XML content is valid.

Note: To add XML headers to calls in IBM® Security AppScan®, add a **Static XML Headers** algorithm on the Request Stack tab of the request.

- Tree

- This view provides a hierarchical view of the XML structure of the message, including elements, namespaces, and the associated values. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit the XML elements and namespaces in the tree.

Use **Skip if Empty** column to select the empty XML elements that you want to skip. This column is visible only if you selected the **Display the 'Skip if Empty' column in XML tree viewer** check box in **Window > Preferences > Test > Test editor > Service test**.

Click **Filter** to hide or show namespace, attribute, or text nodes, depending on your requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable **Allow only valid modifications** if you do not have an XSD or if you want to bypass the schema.

You can right-click an XML element to convert it to an XML fragment. This enables you to perform data correlation (use datapools and create references) on the entire XML fragment instead of only on the value.

- Source

- This view displays the source XML content of the message or plain text content. To format XML content, click **Format XML text**. To wrap XML content into a single line, click **Pack XML text to single line**. Similar controls are available for JSON content. **Important:** In the Source view, do not edit the tags that start with `SoaTag`. If you delete or change these tags, any references and substitutions in

the test will be broken. You cannot recreate these tags after you delete them.

Related reference:

[Stub case details](#)

[Stub response details](#)

Stub case details

Stub case elements enable you to specify the response of a service stub according to the content of an incoming call. You can perform checks on the contents of the XML document of the message return, the number of nodes returned by an XPath query, or the existence of a specific attachment. Each case element has an associated response element. There can be multiple case elements in a stub operation, but the Case : Default element is mandatory.

Default case

The default case contains the default response when no other criteria has been met. When multiple cases are defined, the default case is always the last one to be evaluated.

Contain and equal cases

Contain cases send their response when the incoming call contains the specified XML message. *Equal* cases send their response when the incoming call matched the specified XML message.

The verification occurs if the message return object is a valid XML message. The verification is performed on both the name of the XML element and final return value of the element. Attributes are not checked.

Use the **Form**, **Tree** and **Source** views to edit the message content.

- Test using XML namespace

- Select this option to perform the verification on a qualified structure, including the XML namespace, instead of the simple name. For example, if the expected XML data is:
`<ns1:responseElement xmlns:ns1="http://www.ibm.com/wbse"></ns1:responseElement>`

When **Namespace aware** is enabled, the verification is made on the full name of the return value:
`<ns1:responseElement xmlns:ns1="http://www.ibm.com/wbse"></ns1:responseElement>`

When **Namespace aware** is disabled, the verification ignores the namespace tagging and checks only the simple name of the element and the final return value:
`<ns1:responseElement xmlns:ns1="http://www.ibm.com/wbse"></ns1:responseElement>`

In this case, you can simplify the value of the expected XML data to:
`<responseElement>`
`></responseElement>`

- Test XML text nodes

- Select this option to include XML text values in the verification.

- Test XML attributes

- Select this option to include XML attributes in the verification.

- Form

- This view provides an simple view of the elements of the call with their values. Use this view to quickly edit values in the form.

- Tree

- This view provides a hierarchical view of the elements of the call with their values, attributes, and the associated namespaces. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit this list. Click the **namespace**, **attribute**, or **text** filter buttons, depending on your

requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable smart editing if you do not have an XSD or if you want to bypass the schema.

You can specify standard Java™ regular expressions. In the **Regexp** column, select the line of an attribute or text value and type the regular expression in the **Value** column. For example, the following regular expression checks for a correctly formatted email address: `/^([a-zA-Z0-9_\. \-])+\@(([a-zA-Z0-9 \-])+\.)+([a-zA-Z0-9]{2,4})+$/`

- **Source**

- This view displays the source XML document of the call. **Important:** The ID tags that are shown in the Source page refer to an internal representation for the test. If you remove these tags, you will remove any existing references and substitutions. You cannot re-create these tags after you delete them.

Query case

Query cases send their response when the number of nodes returned by an XML Path language query matches the expected number of nodes specified in the case element.

The verification occurs if the message return object is a valid XML document.

- **XPath expression**

- Specify a query using the XML path language. Refer to the XPath specification for details on expressing an XPath query: <http://www.w3.org/TR/xpath>. Click **Build Expression** to open the XPath Expression Builder window.

Note: Because XPath expressions require that the qualified name has a prefix, XPath expressions will return null for the default namespace declared with `xmlns`.

- **Operator and Expected Count**

- These specify the expected number of nodes returned by the query.

- **Evaluate**

- Click this button to calculate the number of nodes based on the current input. This value automatically replaces the current **Expected count**.

Default case

Attachment verification points return a Pass status when the message return attachment matches all of the criteria specified in the verification point.

The verification occurs only if the message return object is a valid XML document.

- **Enable verification point**

- When selected, the test verifies whether the web service message return objects match the expected criteria of the verification point. An error is reported in the test log if the message return does not match the expected criteria.

- **Index of the attachment to be verified**

- In the case of multiple attachments, this number specifies which attachment to check.

- **Attachment size**

- This specifies the expected size of the attachment.
- **MIME type**
 - This specifies the expected MIME type of the attachment.
- **Encoding**
 - This specifies the expected encoding of the attachment.

Related reference:

[Stub operation details](#)

[Stub response details](#)

Stub response details

In the stub editor, there is one response elements associated with each case element. Stub responses describe the content that is returned by the stub service, simulating the response of the original service.

Message

This page shows the XML content of the request and provides access to data correlation. The same content is presented in three different ways.

- Form

- This view provides a simplified view of the message that focuses on editing the values of the XML content. Use the **Schema** menu to enable assistance with editing XML content so that the XML is valid and complies with the XSD specification. In the Form view, add the XML headers that are required for standard web service calls. On the **Header** bar, click **Add (+)** to create the default XML header structure for WS-Addressing, WS-ReliableMessaging or WS-Coordination requests, or click **More** for other standards. You can enable or disable XML header elements and specify the correct values for each XML element. Checks are performed to ensure that the XML content is valid.

Note: To add XML headers to calls in IBM® Security AppScan®, add a **Static XML Headers** algorithm on the Request Stack tab of the request.

- Tree

- This view provides a hierarchical view of the XML structure of the message, including elements, namespaces, and the associated values. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit the XML elements and namespaces in the tree.

Use **Skip if Empty** column to select the empty XML elements that you want to skip. This column is visible only if you selected the **Display the 'Skip if Empty' column in XML tree viewer** check box in **Window > Preferences > Test > Test editor > Service test**.

Click **Filter** to hide or show namespace, attribute, or text nodes, depending on your requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable **Allow only valid modifications** if you do not have an XSD or if you want to bypass the schema.

You can right-click an XML element to convert it to an XML fragment. This enables you to perform data correlation (use datapools and create references) on the entire XML fragment instead of only on the value.

- Source

- This view displays the source XML content of the message or plain text content. To format XML content, click **Format XML text**. To wrap XML content into a single line, click **Pack XML text to single line**. Similar controls are available for JSON content. **Important:** In the Source view, do not edit the tags that start with `SoaTag`. If you delete or change these tags, any references and substitutions in the test will be broken. You cannot recreate these tags after you delete them.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments. The method for using this identifier depends on your server requirements.

- **MIME or DIME**

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- **Use MTOM transmission mechanism**

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Related reference:

[Stub operation details](#)

[Stub case details](#)

Generic Service Client preferences

The Generic Service Client preferences define the network proxy and authentication credentials to use for importing files from a URL. It also contains the timeout settings.

To access the preferences, click **Window > Preferences > Generic Service Client**. After changing a setting, click **Apply**.

- **Service request timeout**

- Specify the time allocated for the service request by the generic service client

- **URL connection timeout**

- Specify the time allocated to connect to a URL that has a WSDL or XSD file.

- **WSDL parser timeout**

- Specify the time allocated for the WSDL file to be parsed.

- **Proxy host**

- Type the name or IP address of the proxy server.

- **Proxy port**

- Type the port of the proxy server.

- **Keystore**

- Specify the file that includes a certificate to access the server.

- **Keystore password**

- Specify the password for authentication.

- **Service test editor preferences**

The Service test editor preferences control the behavior of the test editor when working with service tests.

- **Service message edition preferences**

The message edition preferences control the behavior of the test editor when editing message content.

- **Service test generation preferences**

The service test generation preferences control how service tests are generated.

- **Raw transaction data view preferences**

The raw transaction data view preferences control how XML data is displayed in the Raw Transaction Data view.

- **Auto values preferences**

The auto values preferences define the default values that are generated by default in SOAP-based service calls.

- **Response time breakdown preferences**

Set these preferences to disable response time breakdown information in service tests. Some server implementations can have problems processing SOAP or HTTP header that include response time breakdown information.

- **Cookies support preferences**

The cookies support preferences define how cookies are managed with HTTP services.

- **WSDL Information Preferences**

The WSDL Information preferences define the tags including server, creation, and technology tags from where certain information is fetched when importing a WSDL

file

Related reference:

[Service test editor preferences](#)

Service test editor preferences

The Service test editor preferences control the behavior of the test editor when working with service tests.

To access the web service test editor preferences, click **Window > Preferences**, expand **Test**, expand **Test Editor**, and click **Web service Test Editor**. After changing a setting, click **Apply**.

- **When response content is XML, ask whether to display content as XML or text.**
 - The test editor can display XML content either as text or as XML. Select this option to be asked how to display the data. By default XML content is displayed as XML.
- **Add quotes around SOAP actions into WSDL model**
 - This option automatically inserts quotes around SOAP actions when this syntax is required by the WSDL.

Related concepts:

[Web service test editor overview](#)

Related tasks:

[Adding elements to a service test](#)

[Comparing service test response contents](#)

Related reference:

[Service test details](#)

[Service message edition preferences](#)

Service message edition preferences

The message edition preferences control the behavior of the test editor when editing message content.

To access the message edition preferences, click **Window > Preferences**, expand **Test**, expand **Service Testing**, and click **Message Edition**. After changing a setting, click **Apply**.

- Editing XML

- These settings specify how XML is displayed in the test editor.

- Background color for source errors

- This setting specifies the color that marks errors in the XML source view.

- Delay used to synchronize source (ms)

- This specifies the number of milliseconds after which the display of the XML source is updated.

- Use color in source view

- This enables colorized XML source on the **Source** tab of web service call elements.

- Create SOAP message with pretty XML serialization

- This improves readability by adding indentation and line wrapping to the XML source displayed in the web service protocol data view.

- Test Contents Tree Label Formatting

- These settings specify how the test elements are displayed in the test contents pane of the test editor.

- Maximum number of parameters displayed in message

- This specifies the number of parameters displayed in the test editor to identify web service test elements. When the number of parameters exceeds this value, only the first parameters are displayed.

- Maximum parameter list displayed in message

- This specifies the number of characters displayed for each parameter in the test editor to identify web service test elements. Parameters that exceed are truncated to the specified value. The minimum value is 3.

- Attachments

- These settings specify the default settings for adding attachments to web service test elements.

- Default MIME type

- This is the default MIME type for new attachments.

- Default encoding

- This is the default encoding for new attachments.

Related reference:

[Service test editor preferences](#)

Service test generation preferences

The service test generation preferences control how service tests are generated. To access the web service test generation preferences, click **Window > Preferences**, expand **Test**, expand **Performance Test**, and click **Web Service Test Generation**. After changing a setting, click **Apply**.

- Time out delay used for call

- This is the default time out delay for service calls. If the service does not respond within this period, an error is produced.

- Time out delay used for call

- This is the default time out delay for asynchronous callbacks. If the service does not respond within this period, the test runs the timeout element of the callback.

- Think time default value

- This is the default think time for generated tests.

- XML data correlation limit

- This is the default maximum number of attributes and text nodes supported for data correlation.

- XML Message maximum length for answers

- This is the default maximum number of characters for the generated XML.

- Text Message maximum length for answers

- This is the default maximum number of characters for the generated text.

- Use case sensitive URL matching

- Select this option to enable the test generation to match URLs from the WSDL with recorded URLs only when their case matches. Disable this option to ignore differences between upper and lower case characters.

Raw transaction data view preferences

The raw transaction data view preferences control how XML data is displayed in the Raw Transaction Data view.

To access the service protocol data view preferences, click **Window > Preferences**, expand **Test**, and click **Raw Transaction Data View**. After changing a setting, click **Apply**.

- **Enable XML source coloring**

- Select this option to enable XML syntax coloring.

- **Coloring styles**

- This section enables you to set a color for each element type in the XML source.

- **Coloring mode**

- This enables you to select the style that is used for start tags, end tags, attributes, and CData tags.

Auto values preferences

The auto values preferences define the default values that are generated by default in SOAP-based service calls.

To access the Auto values preferences, click **Window > Preferences**, expand **Generic Service Client**, and click **Auto Values**. After changing a setting, click **Apply**.

The table lists the default value used for each primitive type. Click **Edit** to modify the default values or click **Reset** to revert to the default settings.

To generate a default value that conforms to the constraints specified in the XSD, select the **Use XSD constraints** check box.

to generate a random default value among those that conform to the regular expression constraints specified in the XSD, select the **Use XSD regular expression constraints** check box.

Related reference:

[Service message edition preferences](#)

Response time breakdown preferences

Set these preferences to disable response time breakdown information in service tests. Some server implementations can have problems processing SOAP or HTTP header that include response time breakdown information.

To access the service protocol data view preferences, click **Window > Preferences**, expand **Test**, and click **Raw Transaction Data View**. After changing a setting, click **Apply**.

- **Include response time breakdown information in SOAP header**

- Specifies whether response time breakdown information is automatically included in the SOAP header of service requests.

- **Include response time breakdown information in HTTP header**

- Specifies whether response time breakdown information is automatically included in the HTTP header of service requests.

Cookies support preferences

The cookies support preferences define how cookies are managed with HTTP services.

To access the cookies support preferences, click **Window > Preferences**, expand **Test**, expand **Service Testing**, and click **Cookies Support**. After changing a setting, click **Apply**.

- **Enable cookies for web services that use HTTP cookies**
 - Select this option to enable support for HTTP cookies in the generic service client. If the option is not selected, cookies are ignored.
- **Load persistent cookies**
 - Select this option to reload cookies that were saved from a previous session. If the option is not selected, cookies are reset when the workbench is restarted.
- **Reload transient cookies**
 - Select this option to reuse saved cookies in existing requests in the generic service client.
- **Cookies specification**
 - Select the specification that is used for supporting cookies. In most cases, use the **best-match** default setting.
- **Release Session Cookies**
 - Click this button to immediately release all saved cookies.

WSDL Information Preferences

The WSDL Information preferences define the tags including server, creation, and technology tags from where certain information is fetched when importing a WSDL file

To access the WSDL information preferences, click **Window > Preferences**, expand **Generic Service Client** and click **WSDL Information**. After changing a setting, click **Apply**.

Web service reports

When you test a web service, these reports are produced during a run and saved after a run.

- **Service Performance report**

The Service Performance report summarizes the validity of the run, summarizes the data most significant to the run, shows the response trend of the slowest 10 service calls in the test, the server health depending on requests, and graphs the response trend of each service calls for a specified interval.

- **Web Service Verification Points report**

The web service verification points report shows the status of the verification points in your tests.

Service Performance report

The Service Performance report summarizes the validity of the run, summarizes the data most significant to the run, shows the response trend of the slowest 10 service calls in the test, the server health depending on requests, and graphs the response trend of each service calls for a specified interval.

Overall page

The Overall page provides the following information:

- A progress indicator that shows the state of the run.
- The bar graph on the left indicates the percentage of successful service calls during the run.
- The bar graph on the right indicates the percentage of verification points with a Pass status for the run.

Summary page

The Summary page summarizes the most important data about the test run, so that you can analyze the final or intermediate results of a test at a glance. The Run Summary table displays the following information:

- The number of virtual users that are active and the number of virtual users that have completed testing. These numbers are updated during the run.
- The elapsed time. This is the total duration of the run, which is displayed in hours, minutes, and seconds.
- The location and name of the test.
- The results for the computer and for all computers. To see summary results for individual computers, click the computer name in the Performance Test Runs view.
- The status of the run. This can be Initializing Computers, Adding Users, Running, Performing Execution History Data Transfer, Stopped, or Complete.
- The total number of virtual users emulated during the test.

The Call Summary section displays the following information:

- The percentage of verification points with a Pass status.
- The total number of verification points with a Fail status.
- The total number of verification points with an Error status.
- The total number of attempted service calls.
- The total number of successful service calls.
- The total number of service calls that produced a timeout.

The Bytes Summary section displays the following information:

- The minimum, maximum, and average number of bytes sent and received for each call in the run.
- The byte rate per second for the run.
- The total number of bytes sent and received for the run.

Response Time Results page

The Response Time Results page shows the average response of the service calls in the test as the test progresses. With this information, you can evaluate system

response during and after the test. The delay between the moment a service call is invoked and the moment the corresponding message return is received, determines the Response times. The bar chart shows the average response time of each service call. Each bar represents a service call that was invoked during the test. As you run the test, the bar chart changes, because the response times are updated dynamically during the run.

The table that follows the bar chart provides the following additional information for each service call:

- The minimum response time during the run.
- The average response time during the run. This matches the information in the chart.
- The maximum response time during the run.
- The standard deviation response time during the run.

Response Time vs. Time Summary page

The Response vs. Time Summary page shows the average response trend as graphed for a specified interval. You set the **Statistics sample interval** value in the schedule, as a schedule property. Measurements that are located in the tests determine the Response times. Response time measurements can be automatically generated between the last input action before a service call and the corresponding message return event.

The line graph shows the average response time for all measurements during the run. Each point on the graph is an average of what has occurred during that interval. The table that follows the graph lists one number: the total average response time for all measurements in the run.

Response Time vs. Time Details page

The Response vs. Time Details page shows the response trend as graphed for a specified interval. You set the **Statistics sample interval** value in the schedule, as a schedule property. The delay between the moment a service call is invoked and the moment the corresponding message return is received determines the Response times.

The line graph shows the average response time of each measurement for a specified interval. A separate line represents each measurement.

The table under the graph provides the following additional information for each response time measurement:

- The minimum response time during the run.
- The average service call response time during the run. This is similar to the graph, but the information in the table includes the entire run.
- The maximum service call response time during the run.
- The standard deviation service call response time during the run.

Data Volume page

The Data Volume page provides details about the volume of data that is sent to and received from the service. You set the **Statistics sample interval** value in the schedule, as a schedule property.

- The **Sent and Received** line graph shows the total bytes sent and received per interval.
- The **Received Summary** table lists, for each call, the received volume rate (bytes per second) for the entire run, the minimum and maximum received bytes per interval, and the average number of bytes received for each call.
- The **Sent Summary** table lists, for each call, the sent volume rate (bytes per second) for the entire run, the minimum and maximum sent bytes per interval, and the average number of bytes sent for each call.

Call Throughput page

The Call Throughput page provides an overview of the frequency of service calls that are being transferred per interval. You set the **Statistics sample interval** value in the schedule, as a schedule property.

- The line graph shows the calls that are started and ended per interval. Ended calls can be: success, fail, or timeout.
- The **Performance Summary** table lists the details of the number of call starts, successes, failures or timeouts for each call and for the run.

Resources page

The Resources page shows all resource counters that are monitored during the schedule run.

- The line chart shows the values of the resources counters monitored during the schedule run. The chart scales automatically to accommodate the highest resource counter value.
- The summary table that follows the chart lists the average values of the resource counters that are monitored during the schedule run. This table is organized by resource monitoring hosts.

Related reference:

[Web Service Verification Points report](#)

Web Service Verification Points report

The web service verification points report shows the status of the verification points in your tests.

Summary page

The Summary page displays a bar graph representing the percentage of successful web service calls for the run. You set the **Statistics sample interval** value in the schedule, as a schedule property.

Below the graph, the Verification Point Summary table lists the following information:

- The percentage of verification points that passed during the run
- The number of verification points that were attempted
- The number of verification points that passed
- The number of verification points that failed

Verification Points Detail page

The Verification Points page displays the details for all types of verification points that were checked during the run.

The Verification Points table lists the following information:

- The number of verification points that passed during the run
- The number of verification points that failed during the run
- The number of verification points that caused an error during the run
- The number of verification points that were inconclusive during the run
- The percentage of verification points that passed during the run

Return Contain Verification Points page

The Return Contain Verification Points page displays the details for contain verification points that were checked during the run.

The table lists the following information:

- The number of verification points that passed during the run
- The number of verification points that failed during the run
- The number of verification points that caused an error during the run
- The number of verification points that were inconclusive during the run
- The percentage of verification points that passed during the run

Return Equal Verification Points page

The Return Equal Verification Points page displays the details for equal verification points that were checked during the run.

The table lists the following information:

- The number of verification points that passed during the run
- The number of verification points that failed during the run
- The number of verification points that caused an error during the run
- The number of verification points that were inconclusive during the run
- The percentage of verification points that passed during the run

Return Query Verification Points page

The Return Query Verification Points page displays the details for query verification points that were checked during the run.

The table lists the following information:

- The number of verification points that passed during the run
- The number of verification points that failed during the run
- The number of verification points that caused an error during the run
- The number of verification points that were inconclusive during the run
- The percentage of verification points that passed during the run

Return Attachment Verification Points page

The Return Attachment Verification Points page displays the details for attachment verification points that were checked during the run.

The table lists the following information:

- The number of verification points that passed during the run
- The number of verification points that failed during the run
- The number of verification points that caused an error during the run
- The number of verification points that were inconclusive during the run
- The percentage of verification points that passed during the run

Related reference:

[Service Performance report](#)

Generic service client overview

The purpose of the generic service client is to send requests to any service that uses an HTTP, JMS, WebSphere® MQ, or Microsoft .NET transport. The generic service client also displays the response returned by the service.

The generic service client is useful for debugging or testing a service when you do not have access to a dedicated client to send the request. You can set up a large variety of transport and security configurations for the service, edit the parameters of the request and send attachments.

When a request is successfully invoked, its message return is added to the **Request History**. You can use this feature to look back at results that were produced at different times.

If you are using IBM® Rational® Performance Tester or IBM Rational Service Tester for SOA Quality, you can select requests in the **Request History** and click **Generate Test** to generate a test that will replay all the selected requests. You can edit the test to replace recorded test values with variable test data, or add dynamic data correlation to the test. You can also set verification points on the contents of the XML documents in the service response.

Supported services

The generic service client enables you to send requests for many types of services that use the following transport protocols:

- HTTP
- Java™ Message Service (JMS), including JBoss and WebSphere implementations
- WebSphere MQ
- Microsoft .NET Framework Windows Communication Foundation (WCF).

Note: If you are using IBM Security AppScan®, only the HTTP transport protocol is supported.

Encryption and security

The Java Runtime Environment (JRE) that the workbench uses must support the level of encryption required by the digital certificate that you select. For example, you cannot use a digital certificate that requires 256-bit encryption with a JRE that supports only 128-bit encryption. By default, the workbench is configured with restricted or limited strength ciphers. To use less restricted encryption algorithms, you must download and apply the unlimited jurisdiction policy files (

`local_policy.jar` and `US_export_policy.jar`).

You can download unlimited jurisdiction policy files from this site:

<http://www.ibm.com/developerworks/java/jdk/security/50/>

Click on **IBM SDK Policy files**, and then log in to developerWorks® to obtain the unlimited jurisdiction policy files. Before installing these policy files, back up the existing policy files in case you want to restore the original files later. Then overwrite the files in `/jre/lib/security/` directory with the unlimited jurisdiction policy files.

SSL Authentication

Service tests support simple or double SSL authentication mechanisms:

- Simple authentication (server authentication): In this case, the test client needs to determine whether the service can be trusted. You do not need to setup a key store. If you select the **Always trust** option, you do not need to provide a server certificate key store.
If you want to really authenticate the service, you can configure an certificate trust store, which contains the certificates of trusted services. In this case, the test will expect to receive a valid certificate.
- Double authentication (client and server authentication): In this case, the service needs to authenticate the test client according to its root authority. You must provide the client certificate keystore that needs to be produced to authenticate the test as a certified client.

When recording a service test through a proxy, the recording proxy sits between the service and the client. In this case, you must configure the SSL settings of the recording proxy to authenticate itself as the actual service to the client (for simple authentication), and as the client to the service (for double authentication). This means that you must supply the recording proxy with the adequate certificates. When using stub services, you can also configure the SSL settings of the stub service to authenticate itself as the actual server. This means that you must supply the service stub with the adequate certificate.

NTLM and Kerberos Authentication

The product supports Microsoft NT LAN Manager (NTLMv1 and NTLMv2) and Kerberos authentication. The authentication information is recorded as part of the test during the recording phase.

To enable NTLMv2 support, you must add a third party library to the workbench. For more information, see [Configuring the workbench for NTLMv2 authentication](#).

Digital certificates

You can test services with digital certificates for both SSL and SOAP security protocol. Digital certificates must be contained in Java Key Store (JKS) keystore resources that are accessible in the workspace. When dealing with keystore files, you must set the password required to access the keys both in the security editor and the test editor. For SOAP security you might have to provide an explicit name for the key and provide a password to access the private keys in the keystore.

Limitations

Arrays are not supported.

Because of a lack of specification, attachments are not supported with the Java Message Service (JMS) transport. The envelope is directly sent using UTF-8 encoding.

All security algorithms are not always available for every Java Runtime Environment (JRE) implementation. If a particular security implementation is not available, add the required libraries to the class path of the JRE that this product uses.

The generic service tester displays the envelope as reflected in the XML document. However, security algorithms consider the envelope as a binary. Therefore, you must set up the SOAP security configuration so that incoming and outgoing messages are correctly encrypted but remain decrypted inside the test.

The Microsoft .NET transport protocol does not support transactions, scopes, or duplex mode requests such as callbacks or two-way services based on the MS-MQ transport.

Related tasks:

- [Sending service requests with WSDL files](#)
- [Sending HTTP endpoint requests](#)
- [Sending a JMS endpoint request](#)
- [Sending a WebSphere MQ endpoint request](#)
- [Testing all operations in a WSDL file](#)
- [Viewing message content](#)
- [Synchronizing a remote WSDL file](#)
- [Adding static XML headers to a service request](#)
- [Opening file attachments](#)
- [Creating SSL configurations](#)
- [Creating a WebSphere MQ transport configuration](#)
- [Creating an HTTP transport configuration](#)
- [Creating a JMS transport configuration](#)
- [Creating Microsoft .NET transport configurations](#)
- [Configuring the environment for SOAP security](#)

Related information:

- [Creating transport protocol configurations](#)

Configuring the environment for service calls

- **Configuring the environment for SOAP security**

SOAP security profiles require access to the libraries that implement encryption, signature, and other security algorithms that transform the XML messages before sending and after receiving them. You must prepare an environment with these libraries to use SOAP security, set the class path of the Java™ Runtime Environment (JRE) that Eclipse uses, and set the class path of the virtual machine that the Agent Controller uses.

Configuring the environment for SOAP security

SOAP security profiles require access to the libraries that implement encryption, signature, and other security algorithms that transform the XML messages before sending and after receiving them. You must prepare an environment with these libraries to use SOAP security, set the class path of the Java™ Runtime Environment (JRE) that Eclipse uses, and set the class path of the virtual machine that the Agent Controller uses.

Before you begin

Before you can test SOAP-based services that use security algorithms, you must obtain a set of security libraries and configuration files for SOAP.

BouncyCastle (<http://www.bouncycastle.org>) is a provider of such security libraries. Use of these security libraries is optional for the Rational® test product.

Procedure

1. Copy the library files into the `jre/lib/ext` of the JRE installation. By default, this is the following directory: `C:\Program Files\IBM\SDP\jdk\jre\lib\ext`
2. Add the following VM argument either to the Eclipse launch command line or to the `eclipse.ini` file: `-vmargs -Dosgi.parentClassLoader=ext` The `eclipse.ini` file is located in the same directory as the `eclipse.exe` launcher binary that is used to run the product.

What to do next

To configure a remote computer that uses only the Agent Controller and does not require access to the workbench, perform only step 1 and restart the Agent Controller service.

After configuring the environment, you must import a Web Services Description Language (WSDL) file and use the WSDL security editor to set up a security profile for the WSDL file.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Sending service requests with WSDL files](#)

[Sending HTTP endpoint requests](#)

Sending service requests with the generic service client

The generic service client enables you to send requests to services for which you do not have a convenient client and to view the responses returned by the service.

- **Generic service client overview**

The purpose of the generic service client is to send requests to any service that uses an HTTP, JMS, WebSphere® MQ, or Microsoft .NET transport. The generic service client also displays the response returned by the service.

- **Creating transport protocol configurations**

- **Sending service requests with WSDL files**

You can send requests to services based on SOAP, Java Messaging Service (JMS), WebSphere MQ, and Microsoft .NET that use a Web Service Description Language (WSDL) file to specify the contents of the service request.

- **Sending HTTP endpoint requests**

You can send requests to services that use an HTTP endpoint.

- **Sending a JMS endpoint request**

You can send requests to services that use a Java™ Messaging Service (JMS) endpoint.

- **Sending a WebSphere MQ endpoint request**

You can invoke calls to services that use a WebSphere MQ endpoint.

- **Testing all operations in a WSDL file**

You can use the generic service client to rapidly send requests to a service using all the operations in a Web Services Description Language (WSDL) file. The calls are generated with default values based on the type of data.

- **Viewing message content**

The Raw Transaction Data view displays the raw XML, text, or binary content of any service request or response that is selected in the generic service client.

- **Synchronizing a remote WSDL file**

For web services that make their Web Services Description Language (WSDL) file available from a URL, you might have to ensure that the WSDL that you work with is always up to date. By synchronizing the WSDL, you ensure that the local copy of the WSDL in your workspace is regularly synchronized with the remote WSDL.

- **Adding static XML headers to a service request**

You can add static XML headers to service requests to ensure compliance with WS-Addressing, WS-ReliableMessaging, and WS-Coordination specifications as well as other predefined standards.

- **Opening file attachments**

When a service sends a file attachment with the response, you must import it as a resource to open the attachment.

Creating transport protocol configurations

- **Creating an HTTP transport configuration**

You can create an HTTP transport configuration that describes the transport settings for a service request. Transport and security settings can be associated with any service request.

- **Configuring the workbench for NTLMv2 authentication**

NTLMv2 authentication requires access to a third-party library. To record and execute a test that contains NTLMv2 authentication, you must download the library and place it at the right location.

- **Creating a JMS transport configuration**

You can create an JMS transport configuration that describes the transport settings for a service request that uses the Java Message Service (JMS) protocol, including JBoss and IBM® WebSphere JMS. Transport and security settings can be associated with any service request.

- **Creating a WebSphere MQ transport configuration**

You can create a transport configuration that describes the transport settings for a service request that uses the IBM WebSphere MQ protocol. Transport and security settings can be associated with any service request.

- **Creating Microsoft .NET transport configurations**

You can manually create a Microsoft .NET transport configuration to describe the transport settings for service requests that use the Windows Communication Foundation (WCF) protocol.

- **Creating SSL configurations**

You can create a Secure Sockets Layer (SSL) configuration that describes the settings for a service request that uses SSL certification mechanisms. SSL configurations can be associated with any service request that uses the HTTP or IBM WebSphere MQ transport protocols.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Sending service requests with WSDL files](#)
[Sending HTTP endpoint requests](#)
[Sending a JMS endpoint request](#)
[Sending a WebSphere MQ endpoint request](#)
[Testing all operations in a WSDL file](#)
[Viewing message content](#)
[Synchronizing a remote WSDL file](#)
[Adding static XML headers to a service request](#)
[Opening file attachments](#)

Creating an HTTP transport configuration



You can create an HTTP transport configuration that describes the transport settings for a service request. Transport and security settings can be associated with any service request.

Before you begin

If you are using Secure Sockets Layer (SSL) authentication, ensure that you have valid key files in your workspace.

If you are using SOAP security, ensure that you have configured the environment with the correct libraries and configuration files.

Procedure

1. Click the **Generic service client**  toolbar button to open the generic service client and click the **Transport** tab. This opens the Transport Configurations page.
2. On the Transport Configurations page, click **Create an HTTP configuration**  to create a new HTTP transport configuration.
3. Type a **Name** for the new transport configuration.
4. Specify the following options for the HTTP transport:
 - **Use HTTP Keep Alive**
 - Select this option to keep the HTTP connection open after the request. This option is not available if you are using IBM® Rational® AppScan®.
 - **Use SSL**
 - Select this option to use an SSL configuration. Click **Configure SSL** to create an SSL configuration or select an existing configuration.
 - **Platform Authentication**
 - In this section, specify the type of authentication that is required to access the service. Select **None** if no authentication is required.
 - **Basic HTTP authentication**
 - Select this option to specify the **User Name** and **Password** that are used for basic authentication.
 - **NTLM authentication**
 - Select this option to use the Microsoft NT LAN Manager (NTLM) authentication protocol. NTLM uses challenge-response authentication. This view lists what is negotiated (supported by the client and requested of the server) and what is authenticated (the client reply to the challenge from the server).
 - **Kerberos authentication**
 - Select this option to use the Kerberos authentication protocol between the client and server.
 - **Connect through proxy server**
 - If the HTTP connection needs to go through a proxy server or a corporate firewall, specify the **Address** and **Port** of the proxy server. If the proxy requires authentication, select either **Basic proxy authentication** or **NTLM proxy authentication**.
 - **Proxy authentication**

- In this section, specify the type of authentication that is required to access the proxy. Select **None** if no authentication is required.
 - **Basic proxy authentication**
 - Select this option to specify the **User Name** and **Password** that are used for basic authentication.
 - **NTLM proxy authentication**
 - Select this option to use the Microsoft NT LAN Manager (NTLM) authentication protocol. NTLM uses challenge-response authentication. This view lists what is negotiated (supported by the client and requested of the server) and what is authenticated (the client reply to the challenge from the server).
- **Custom class**
 - Select this option if the communication protocol requires complex, low-level processing with a custom Java™ code to transform incoming or outgoing messages. Click **Browse** to select a Java class that uses the corresponding API. This option is not available in IBM Security AppScan.

See [Creating SSL configurations](#) for more information about SSL authentication.

5. Click **OK** to create the new configuration.

What to do next

Once created, you can use your new configuration with any service request that uses the HTTP transport protocol. You can use the **Configurations** list in the generic service client to edit existing configurations or to create duplicate configurations.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Creating SSL configurations](#)

[Creating a WebSphere MQ transport configuration](#)

[Creating a JMS transport configuration](#)

[Creating Microsoft .NET transport configurations](#)

[Sending HTTP endpoint requests](#)

Configuring the workbench for NTLMv2 authentication

NTLMv2 authentication requires access to a third-party library. To record and execute a test that contains NTLMv2 authentication, you must download the library and place it at the right location.

Before you begin

Before you can test SOAP-based services that use security algorithms, you must obtain and install a third-party library file.

About this task

By default, the HTTP test generation does not enable NTLMv2 authentication, even if it was part of the recording. To automatically enable the correct NTLM version from the recording, set the NTLM V2 setting to **Automatic** in the HTTP Test Generation preferences.

Procedure

To configure the workbench to enable NTLMv2 authentication

1. Download the archive from <http://jcifs.samba.org/src/jcifs-1.3.18.zip>
2. Unarchive the zip file and copy the JAR file to the installation directory:
`InstallationDirectory\plugins\com.ibm.rational.test.lt.provider_<version>`
3. To automatically enable the correct NTLM version from the recording, In the workbench, click **Window > Preferences > Test > HTTP Test Generation** and set the **NTLM v2** setting to **Automatic**.

Results

When a test was recorded with NTLMv2, the **NTLM V2** setting is selected in the test editor, under NTLM Authentication.





Creating a JMS transport configuration

You can create an JMS transport configuration that describes the transport settings for a service request that uses the Java™ Message Service (JMS) protocol, including JBoss and IBM® WebSphere® JMS. Transport and security settings can be associated with any service request.

Before you begin

If you are using SOAP security, ensure that you have configured the environment with the correct libraries and configuration files.

Procedure

1. Click the **Generic service client**  toolbar button to open the generic service client and click the **Transport** tab. This opens the Transport Configurations page.
2. On the Transport Configurations page, click one of the following buttons:
 - **Create a basic JMS configuration** () to create a new generic JMS transport configuration.
 - **Create a JBoss JMS configuration** () to create a JMS configuration preconfigured for JBoss.
 - **Create a WebSphere JMS configuration** () to create a JMS configuration preconfigured for WebSphere JMS.
3. Type a **Name** for the new transport configuration and select whether the service is a **queue** or a **topic** destination.
4. Type the address of the JMS end point.
5. Select **Use temporary object** to provide the address of the reception point to the service as a temporary object. If you disable this setting, you must manually specify the reception point address.
6. If the service requires authentication, select **Basic Authentication** and type the user name and password to access the service.
7. If the service requires a custom Java Naming and Directory Interface (JNDI) adapter, you can provide your own Java class that extends the Apache Axis class. In this case, select Custom Adapter and specify the name of the custom Java class.
8. Specify whether the message type is **Text** or **Binary**.
9. If necessary, click **Add** or **Edit** to specify the **Context factory properties** or **Connector properties** required to access the service.
10. Click **OK** to create the new configuration.

What to do next

Once created, you can use your new configuration with any service request that uses the JMS transport protocol. You can use the **Configurations** list in the generic service client to edit existing configurations or to create duplicate configurations.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Creating SSL configurations](#)

[Creating a WebSphere MQ transport configuration](#)

[Creating an HTTP transport configuration](#)

[Creating Microsoft .NET transport configurations](#)

[Sending a JMS endpoint request](#)



Creating a WebSphere MQ transport configuration

You can create a transport configuration that describes the transport settings for a service request that uses the IBM® WebSphere® MQ protocol. Transport and security settings can be associated with any service request.

Before you begin

If you are using SOAP security, ensure that you have configured the environment with the correct libraries and configuration files.

Procedure

1. Click the **Generic service client**  toolbar button to open the generic service client and click the **Transport** tab. This opens the Transport Configurations page.
2. On the Transport Configurations page, click **Create a WebSphere MQ configuration**  to create a new generic MQ transport configuration.
3. Type a **Name** for the new transport configuration and select whether the service is a **queue** or a **topic** destination.
4. Specify the **Queue Manager Name** for the queue manager that will receive the call, and the **Queue Name** for the queue managed by the queue manager.
5. If the WebSphere MQ server is running on the local computer, select **Local Queue Manager**. If not, specify the **Address**, **Port**, and **Client Channel** for the remote WebSphere MQ server.
6. If you want the server to create a temporary queue for receiving messages, select **Use Temporary Queue for Response**. If not, specify the queue (handled by the specified queue manager) that will receive responses from the WebSphere MQ server.
7. If you are using the Microsoft .NET framework with SOAP over MQ, specify the name of the target service.
8. If you are using SOAP over MQ, select **Use RFH2 header**. Otherwise, specify the **Message Descriptor** and **Encoding** options for the message header.
9. If the service requires SSL authentication, click **Configure SSL** to select an existing SSL configuration or to create a new one. See [Creating SSL configurations](#).
10. Click **OK** to create the new configuration.

What to do next

Once created, you can use your new configuration with any service request that uses the WebSphere MQ transport protocol. You can use the **Configurations** list in the generic service client to edit existing configurations or to create duplicate configurations.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Creating SSL configurations](#)

[Creating an HTTP transport configuration](#)

[Creating a JMS transport configuration](#)

[Creating Microsoft .NET transport configurations](#)

[Sending a WebSphere MQ endpoint request](#)

Creating Microsoft .NET transport configurations

You can manually create a Microsoft .NET transport configuration to describe the transport settings for service requests that use the Windows Communication Foundation (WCF) protocol.

Before you begin

If you are using SOAP security, ensure that the environment is configured with the correct libraries and configuration files.

Certificates and libraries required by the Microsoft client proxy must be installed on the computer, including Microsoft .NET libraries.

You must link a modified version of the Microsoft client proxy configuration file of the WCF service (by default `client.exe.config`) to the Microsoft .NET transport configuration. You must rename the file to `soaclient.exe.config` and edit it as described in the following procedure.

Tip: You can create a Microsoft .NET transport configuration automatically by importing the Microsoft .NET WSDL file. In this case, you must still manually edit the Microsoft .NET transport configuration to point to the modified `soaclient.exe.config` file as described in the following procedure. For more information, see [Sending service requests with WSDL files](#)

About this task

The product supports testing WCF services that use the following bindings:

- BasicHttpBinding
- WsHttpBinding
- NetMsMqBinding for 1-way calls only
- WSFederationHttpBinding
- WS2007FederationHttpBinding
- NetTcpBinding
- Custom bindings that do not integrate custom extensions in the channel, serialization of the message, transport, and security

Note: The following WCF services are not supported:

- Transaction and scopes
- Duplex mode requests, such as callbacks or 2-way services based on the Microsoft Message Queuing (MS-MQ) transport

Procedure

1. Create a modified `soaclient.exe.config` file by completing the following steps:
 - A. Create a copy of `client.exe.config` (or `proxy_client_name.config`) file from the Microsoft .NET project and rename the copy to `soaclient.exe.config`.
 - B. Edit the `soaclient.exe.config` file to use the version of Microsoft .NET that the product supports, as specified on the following line:`<supportedRuntime version="v4.0"`


```
sku=".NETFramework,Version=v4.0"/>
```


C. Edit the `soaclient.exe.config` file so that the endpoints in the configuration file point to the client contract of the product, as specified on the following line:

```
contract="IBM.ServiceModel.Soa.Extension.Stub.IStubTest"
```

D. Import the modified `soaclient.exe.config` file into the workspace.

After you create the `soaclient.exe.config` file, you can skip the following steps and import the WSDL file to automatically create a Microsoft .NET transport configuration based on the information provided by the WSDL. For more information, see [Sending service requests with WSDL files](#).

2. Click the **Generic service client** toolbar button () to open the generic service client and click the **Transport** tab.
3. On the Transport Configurations page, click **Create a Microsoft .NET configuration**.
4. Type a name for the new transport configuration and specify the following options:
 - **Location of soaclient.exe.config**
 - Specify the location of the `soaclient.exe.config` file. You must create this file manually by copying and editing the `client.exe.config` file from the Microsoft .NET service.
 - **User authentication**
 - If the service requires authentication, select **User Authentication** and type the user name and password to access the service.
 - **Endpoint protection**
 - By default, the transport configuration uses the endpoint protection level that is described in the `soaclient.exe.config` file. Use this setting to specify a different **Protection level**:
 - **Signature**: Select this option to digitally sign requests.
 - **Encryption and Signature**: Select this option to digitally sign and encrypt requests.
 - **Advanced properties**
 - Use this table to list the request and response actions by order of the methods in the WSDL file. Click **Add** to specify the name and value of request and response actions that are required by the service. This table is generated automatically when you import a Microsoft .NET WSDL file.
5. Click **OK** to create the transport configuration.

What to do next

After you create the configuration, you can use it with any service call that uses the Microsoft .NET transport protocol. You can use the **Configurations** list in the generic service client to edit existing configurations or to create duplicate configurations.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Creating SSL configurations](#)

[Creating a WebSphere MQ transport configuration](#)

[Creating an HTTP transport configuration](#)

Creating a JMS transport configuration

Creating SSL configurations

You can create a Secure Sockets Layer (SSL) configuration that describes the settings for a service request that uses SSL certification mechanisms. SSL configurations can be associated with any service request that uses the HTTP or IBM® WebSphere® MQ transport protocols.

Before you begin




If you are using SSL, ensure that you have valid certificate keystore files in your workspace.

If you are using SOAP security, ensure that you have configured the environment with the correct libraries and configuration files. See [Configuring the environment for SOAP security](#) for more information.

About this task

If you have to use different mutual SSL authentications for virtual testers in a test, you can create a datapool that stores all of the trust aliases names. In the test editor, in the **SSL Configuration** tab, you add a SSL configuration and associate it with the datapool. When a schedule is run, the SSL configuration is applied to each virtual tester.

Procedure

1. Click the **Generic service client**  toolbar push button to open the generic service client, and click the **Transport** tab.
2. Either open an existing HTTP or WebSphere MQ transport configuration, or create a new one, and then click **Configure SSL**.
3. Click **Rename**  to rename the default SSL configuration or **New**  to create one.
4. Specify the following settings for the SSL configuration.
 - **Server Authentication**
 - This section describes how the client trusts the server.
 - **Always trust server**
 - Select this option if no authentication is required or to ignore server certificates so that all servers are trusted. If you are using single authentication and you want to accept trusted servers only, then disable this option and specify a truststore that contains the trusted server certificates.
 - **Client truststore**
 - When you are using single authentication, the client truststore contains the certificates of all trusted servers. Click **Browse** to specify a KS, JKS, or JCEKS file containing valid certificates of the trusted servers.
 - **Password**
 - If the client truststore file is encrypted, type the password required to access the file.
 - **Mutual Authentication**

- This section describes how the server trusts the client in addition to server authentication.

- **Use client-side certificate**

- If you are using double authentication, select this option to specify a keystore containing the client certificate. This certificate allows the server to authenticate the client.

- **Client certificate keystore**

- Click **Browse** to specify a KS, JKS, or JCEKS file containing a valid certificate that authenticates the client.

- **Password**

- If the client truststore file is encrypted, type the password required to access the file.

- **Select trust alias for Mutual Authentication**

- Select an alias to be used for the SSL configuration. There could be multiple aliases in a keystore for different security certificates. Choose an appropriate alias for a user. You can also use datapool to store aliases that you can apply to virtual users at run time.

Note: You can copy the contents from an SSL configuration into another SSL configuration by using **Copy**  and **Paste**  in the SSL editor.

5. Click **OK** to create the configuration, and close the SSL editor.

What to do next

When the SSL configuration is created, you can use the SSL configuration with any service request that uses SSL certification. You can use the SSL editor to edit existing configurations.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Creating a WebSphere MQ transport configuration](#)

[Creating an HTTP transport configuration](#)

[Creating a JMS transport configuration](#)

[Creating Microsoft .NET transport configurations](#)

Sending service requests with WSDL files

You can send requests to services based on SOAP, Java Messaging Service (JMS), WebSphere® MQ, and Microsoft .NET that use a Web Service Description Language (WSDL) file to specify the contents of the service request.

Before you begin

Ensure that you have a valid WSDL file, which is accessible either on the file system, in the workspace, at a specific URL, or in an IBM® WebSphere Service Registry and Repository or a Universal Description Discovery and Integration (UDDI) repository.

Ensure that the WSDL files use the correct syntax for the test environment. The generic service client might not work with some WSDL files.

If the service uses Secure Sockets Layer (SSL) authentication, create an SSL configuration before sending the request. For more information, see [Creating SSL configurations](#).

If the service uses SOAP security for encryption, signature, or other security algorithms, you must first configure the environment with the correct libraries and configuration files, and then create a WSDL security profile. For more information, see [Configuring the environment for SOAP security](#) and [Creating security profiles for WSDL files](#).



To import a WSDL file from a secured site that requires mutual authentication, you must have the Keystore file in the workspace.

About this task



When you create a call from a WSDL file, the call is configured automatically with any SOAP, JMS, WebSphere MQ, or Microsoft .NET endpoints that are available in the WSDL file. Select the corresponding transport configuration on the Transport page of the request. **Note:** For the specific requirements related to Microsoft .NET support, see [Creating Microsoft .NET transport configurations](#).

Procedure

To send a service request based on a WSDL file:

1. Click the **Open the Generic Service Client** toolbar button  and select the Requests page.
2. Click **Add+** and select the method to add a WSDL file or click the corresponding shortcut button on the main page.
 - Click **Add WSDL from Workspace** to add a WSDL file from the local workspace.
 - Click **Add WSDL from File System** to add a WSDL file from the file system.
 - Click **Add WSDL from URL** to download and import an online WSDL from the web.
 - Click **Add WSDL from WSRR** to add a WSDL from WebSphere Service Registry and Repository. Enter the URL of the WebSphere Service Registry and Repository and click **Connect**. You can click **Search**  to browse the contents of


the repository.

- Click **Add WSDL from UDDI** to add a WSDL from a Universal Description Discovery and Integration (UDDI) repository. Enter the URL of the UDDI and click **Connect**. You can click **Filter**  and **Search**  to browse the contents of the repository.

Note: If you are importing the WSDL file from a secured site that requires certificate authentication, click **Import Properties** and, for **Keystore**, select the keystore file that contains the certificate to be provided to the server, and for the **Keystore password**, type the password.

3. Click **OK**. The WSDL file is added to the Request Library.
 4. In the Request Library, expand the WSDL file, binding, and operation, and then select the call element. The generic service client shows three steps: **Edit Data**, **Invoke** and **View Response**. The details for the call are displayed under the **Edit Data** step.
 5. On the Message page, use the Form, Tree, or Source views to edit the contents of the request. Each view shows a different format of the same data. To add or remove XML elements in the Form or Tree view, click **Schema > Validate and Assist** to comply with an XML Schema Definition (XSD) specified in the schema catalog.
 6. On the Transport page, specify whether to use an HTTP, JMS, WebSphere MQ, or Microsoft .NET transport configuration for the request. The transport information from the WSDL file is imported automatically into the transport configuration. For Microsoft .NET, select the corresponding transport configuration and specify the location of the `soaclient.exe.config` file. You must create this file manually. For details, see [Creating Microsoft .NET transport configurations](#).
- Note:** If you are using IBM Security AppScan®, only the HTTP transport protocol is available.
7. On the Request Stack page, specify whether to override the security or processing algorithms that are applied to the outgoing request for the WSDL file. Click **Show Response Stack** to add a Response Stack page to edit the security or processing algorithms for incoming responses. **Note:** These settings apply only to the current request. If you want to edit the request or response stack for all requests that use the current WSDL file, click **Edit WSDL Security** to open the WSDL Security Editor.
 8. When you are ready to send the service request, click **Invoke**. The generic service client sends the request and displays the message return under the **View Response** step.

What to do next

Successful requests are recorded and added to the **Request History** list. If you are using IBM Rational® Performance Tester or IBM Rational Service Tester for SOA Quality, you can click the **Generate Test Suite** button  to create a service test.

Related concepts:

[Generic service client overview](#)

Related tasks:

- [Sending HTTP endpoint requests](#)
- [Sending a JMS endpoint request](#)
- [Sending a WebSphere MQ endpoint request](#)
- [Testing all operations in a WSDL file](#)
- [Viewing message content](#)
- [Synchronizing a remote WSDL file](#)
- [Adding static XML headers to a service request](#)
- [Opening file attachments](#)
- [Configuring the environment for SOAP security](#)

Related information:

- [Creating transport protocol configurations](#)

Sending HTTP endpoint requests

You can send requests to services that use an HTTP endpoint.



Before you begin

If the service uses Secure Sockets Layer (SSL) authentication, create an SSL configuration before sending the request. For more information, see [Creating SSL configurations](#).

If the service uses SOAP security for encryption, signature, or other security algorithms, you must first configure the environment with the correct libraries and configuration files, and then create a security profile for the WSDL file. For more information, see [Configuring the environment for SOAP security](#) and [Creating security profiles for WSDL files](#)

Procedure


To send a request to an HTTP service:

1. Click the **Open the Generic Service Client** toolbar button  and select the Requests page.
2. Click the **Add** icon  and click a type of request that you want to send or in Request Library, right-click **EndPoints** and select a type of request that you want to send.
3. In the Configure Protocol window, select **HTTP** and specify the HTTP transport configuration. If necessary, click **New** to create an HTTP transport configuration for the call.
4. Type the URL of the call, the HTTP method and version, and specify any header or cookie properties. Click the **Rest mode** check box to split the URL into resource and parameters.
5. Click **Next**.
6. On the Select Root Element page, if the service uses a specific XML Schema Definition (XSD), select one from the list or click **Browse** to import the XSD file, and then, select the root element for the request. If no XSD is available for the service, select **No Schema**.
7. Click **Finish**. The request is added to the **Endpoints** section of the Request Library.
8. In the Request Library, select the request element. The generic service client shows three steps: **Edit Data**, **Invoke**, and **View Response**. The details for the call are displayed under the **Edit Data** step.
9. On the Message page, use the Form, Tree, or Source views to edit the contents of the request. Each view shows a different format of the same data. To add or remove XML elements in the Form or Tree view, click **Schema > Validate and Assist** to comply with an XSD specified in the schema catalog.
10. On the Attachments page, specify any file attachments to send with the request. To add an attachment, click **Add** and follow the wizard to attach a file with the request.
11. On the Transport page, if necessary, change the transport configuration for the request. You can create and edit transport and security configurations by

clicking the **Transport** tab.

12. On the Request Stack page, specify whether to override the security or processing algorithms that are applied to the outgoing request for the WSDL. Click **Show Response Stack** to add a Response Stack page to edit the security or processing algorithms for incoming responses. **Note:** These settings apply only to the current request. To edit the request or response stack for all requests that use the current WSDL file, click **Edit WSDL Security** to open the WSDL Security Editor.
13. When you are ready, click **Invoke** to send the service request. The generic service client sends the request and displays the message return under the **View Response** step.

What to do next

Successful requests are recorded and added to the **Request History** list. If you are using IBM Rational® Performance Tester or IBM Rational Service Tester for SOA Quality, you can click the **Generate Test Suite** button () to create a service test.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Sending service requests with WSDL files](#)
[Sending a JMS endpoint request](#)
[Sending a WebSphere MQ endpoint request](#)
[Testing all operations in a WSDL file](#)
[Viewing message content](#)
[Synchronizing a remote WSDL file](#)
[Adding static XML headers to a service request](#)
[Opening file attachments](#)
[Configuring the environment for SOAP security](#)
[Creating an HTTP transport configuration](#)

Related information:

[Creating transport protocol configurations](#)

Sending a JMS endpoint request


You can send requests to services that use a Java™ Messaging Service (JMS) endpoint.

Before you begin

If the service uses Secure Sockets Layer (SSL) authentication, create an SSL configuration before sending the request. For more information, see [Creating SSL configurations](#).


Procedure

To send a request to a JMS service:

1. Click the **Open the Generic Service Client** toolbar button () and select the **Requests** page.
2. Click **Add** (+) and click a type of request that you want to send or in Request Library, right-click **EndPoints** and select a type of request that you want to send.
3. In the Configure Protocol window, select **JMS** and specify the JMS transport configuration. If necessary, click New to create an JMS transport configuration for the call.
4. Click **Add** to specify any properties that are to be sent with the call.
5. Click **Next**.
6. On the Select Root Element page, if the service uses a specific XML Schema Definition (XSD), select one from the list or click **Browse** to import the XSD file, and then, select the root element for the call. If no XSD is available for the service, select **No Schema**.
7. Click **Finish**. The request is added to the **Endpoints** section of the Request Library.
8. In the Request Library, select the request element. The generic service client shows 3 steps: **Edit Data**, **Invoke**, and **View Response**. The details for the request are displayed under the **Edit Data** step.
9. On the Message page, use the **Form**, **Tree**, or **Source** views to edit the contents of the request. Each view shows a different format of the same data. If you want to add or remove XML elements in the Form or Tree view, you can click **Schema > Validate and Assist** to comply with an XSD specified in the schema catalog.
10. On the Transport page, if necessary, change the transport configuration for the request. You can create and edit transport and security configurations by clicking the **Transport** tab.
11. On the Request Stack page, specify whether you want to override the security or processing algorithms that are applied to the outgoing request for the WSDL. Click **Show Response Stack** to add a Response Stack page to edit the security or processing algorithms for incoming responses. **Note:** These settings apply only to the current request. If you want to edit the request or response stack for all requests that use the current WSDL file, click **Edit WSDL Security** to open the WSDL Security Editor.

12. When you are ready, click **Invoke** to send the service request. The generic service client sends the request and displays the message return under the **View Response** step.

What to do next

Successful requests are recorded and added to the **Request History** list. If you are using IBM Rational® Performance Tester or IBM Rational Service Tester for SOA Quality, you can click the **Generate Test Suite** button () to create a service test.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Sending service requests with WSDL files](#)
[Sending HTTP endpoint requests](#)
[Sending a WebSphere MQ endpoint request](#)
[Testing all operations in a WSDL file](#)
[Viewing message content](#)
[Synchronizing a remote WSDL file](#)
[Adding static XML headers to a service request](#)
[Opening file attachments](#)
[Creating a JMS transport configuration](#)

Related information:

[Creating transport protocol configurations](#)

Sending a WebSphere MQ endpoint request

You can invoke calls to services that use a WebSphere® MQ endpoint.


Before you begin

If the service uses Secure Sockets Layer (SSL) authentication, create an SSL configuration before sending the request. For more information, see [Creating SSL configurations](#).

If the service uses SOAP security for encryption, signature, or other security algorithms, you must first configure the environment with the correct libraries and configuration files, and then create a security profile for the WSDL file. For more information, see [Configuring the environment for SOAP security](#) and [Creating security profiles for WSDL files](#).

Procedure


To send a request to an WebSphere MQ service:

1. Click the **Open the Generic Service Client** toolbar button () and select the **Requests** page.
2. Click **Add** (+) and click a type of request that you want to send or in Request Library, right-click **EndPoints** and select a type of request that you want to send
3. In the Configure Protocol window, select **WebSphere MQ** and specify the WebSphere MQ transport configuration. If necessary, click **New** to create an WebSphere MQ transport configuration for the call.
4. Specify the SOAP action. If the service requires that you override the header specified in the WebSphere MQ transport configuration, select **Override MQ protocol configuration values** and specify the correct details.
5. Click **Next**.
6. On the Select Root Element page, if the service uses a specific XML Schema Definition (XSD), select one from the list or click **Browse** to import the XSD file, and then, select the root element for the request. If no XSD is available for the service, select **No Schema**.
7. Click **Finish**. The request is added to the **Endpoints** section of the Request Library.
8. In the Request Library, select the request element. The generic service client shows three steps: **Edit Data**, **Invoke**, and **View Response**. The details for the request are displayed under the **Edit Data** step.
9. On the Message page, use the Form, Tree, or Source views to edit the contents of the request. Each view shows a different format of the same data. To add or remove XML elements in the Form or Tree view, click **Schema > Validate and Assist** to comply with an XSD specified in the schema catalog.
10. On the Transport page, if necessary, change the transport configuration to be used by the request. You can create and edit transport and security configurations by clicking the **Transport** tab.
11. On the Request Stack page, specify whether you want to override the security or processing algorithms that are applied to the outgoing request for the WSDL file. Click **Show Response Stack** to add a Response Stack page to edit the security

or processing algorithms for incoming responses. **Note:** These settings apply only to the current request. To edit the request or response stack for all requests that use the current WSDL file, click **Edit WSDL Security** to open the WSDL Security Editor.

12. When you are ready, click **Invoke** to send the service request. The generic service client sends the request and displays the message return under the **View Response** step.

What to do next

Successful requests are recorded and added to the **Request History** list. If you are using IBM Rational® Performance Tester or IBM Rational Service Tester for SOA Quality, you can click the **Generate Test Suite** button () to create a service test.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Sending service requests with WSDL files](#)
[Sending HTTP endpoint requests](#)
[Sending a JMS endpoint request](#)
[Testing all operations in a WSDL file](#)
[Viewing message content](#)
[Synchronizing a remote WSDL file](#)
[Adding static XML headers to a service request](#)
[Opening file attachments](#)
[Creating a WebSphere MQ transport configuration](#)

Related information:

[Creating transport protocol configurations](#)

Testing all operations in a WSDL file

You can use the generic service client to rapidly send requests to a service using all the operations in a Web Services Description Language (WSDL) file. The calls are generated with default values based on the type of data.





Before you begin

Ensure that you have a valid WSDL file. Ensure that the WSDL files use the correct syntax for the test environment. The generic service client might not work with some Web Services Description Language (WSDL) files.


If the service uses Secure Sockets Layer (SSL) authentication, create an SSL configuration before invoking the call. See [Creating SSL configurations](#) for details. If the service uses SOAP security for encryption, signature, or other security algorithms, you must first configure the environment with the correct libraries and configuration files, and then create a security profile for the WSDL. See [Configuring the environment for SOAP security](#) and [Creating security profiles for WSDL files](#) for details.

Calls will be generated for each operation in the WSDL file using the default values for each type. For example, strings will use the default value `str`. You can change the default values in the XML Default Values preferences.

Procedure

1. Open the generic service client and click the **Requests** tab, and then, click  **Add a WSDL file**.
2. In the Add WSDL Files window, select an existing WSDL or import a WSDL with one of the following methods:
 - Click **Import from File** to import a WSDL file from the file system.
 - Click **Import from URL** to download and import an online WSDL from the web.
 - Click **Import from WSRR** to import a WSDL from an IBM WebSphere Service Registry and Repository (WSRR). Enter the URL of the WSRR and click **Connect**. You can click  **Search** to browse the contents of the repository.
 - Click **Import from UDDI** to import a WSDL from a Universal Description Discovery and Integration (UDDI) repository. Enter the URL of the UDDI and click **Connect**. You can click  **Filter** and  **Search** to browse the contents of the repository.
3. Click **OK**. The WSDL is added to the Call Library.
4. In the Call Library, right-click the WSDL and select **Test WSDL Methods**. The call is automatically configured with any SOAP or JMS endpoints that are available in the WSDL.

What to do next

Successful calls are recorded and added to the **Request History** list. If you are using IBM Rational® Performance Tester or IBM Rational Service Tester for SOA Quality, you can click the **Generate Test Suite** () button to create a service test.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Sending service requests with WSDL files](#)

[Sending HTTP endpoint requests](#)

[Sending a JMS endpoint request](#)

[Sending a WebSphere MQ endpoint request](#)

[Viewing message content](#)

[Synchronizing a remote WSDL file](#)

[Adding static XML headers to a service request](#)

[Opening file attachments](#)

Related reference:

[Auto values preferences](#)

Related information:

[Creating transport protocol configurations](#)

Viewing message content

The Raw Transaction Data view displays the raw XML, text, or binary content of any service request or response that is selected in the generic service client.

About this task

The Raw Transaction Data view displays plain text, XML, or binary data, depending on the type of the message content.

Procedure

To view text, XML, or binary message content:

1. In the generic service client, click the **View** menu, and select **Raw Transaction Data**. If you are using IBM® Rational® Performance Tester or IBM Rational Service Tester for SOA Quality, click **Window > Show View > Other > Generic Service Client > Raw Transaction Data**
2. Select a service request or response. If you are using IBM Rational Performance Tester or IBM Rational Service Tester for SOA Quality, this view is also linked to the selected request or response in service tests, service stubs, or in the test log.
3. Depending on the nature of the message content, the following actions are available:
 - **Text mode**
 - When a plain text element is displayed, you can select and copy text. Click **Colorize Text** to enable or disable text colorization for HTML.
 - **XML mode**
 - When an XML element is displayed, you can select and copy text. Click **Colorize Text** to enable or disable text colorization for XML. Click **Enable XML Pretty Serialization** to improve readability by adding line breaks and indentation to the XML content.
If the XML content is modified by a request or response stack or by the WSDL security editor, the Stack Contents pane displays the list of steps in the stack. You can select each step to view the changes to the XML content. You can also select one or two steps and click **Compare Steps** to open a comparison window.
 - **Binary mode**
 - When a binary element is displayed, you can switch between **Binary** and **Raw-ASCII** views. Right-click the binary view to perform the following actions:
 - **Select**: Opens the Select window, where you can select binary data by string or by specifying the number of characters to select. When a portion of binary data is selected, you can copy it to the clipboard.
 - **Go to Offset**: Opens the Go to Offset window, where you can move to bytes at a particular offset.
 - **Find**: Opens the Find window, where you can search for and replace binary data in a number of formats.
 - **Encodings**: Select the encoding to use for displaying binary data in the text column.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Sending service requests with WSDL files](#)

[Sending HTTP endpoint requests](#)

[Sending a JMS endpoint request](#)

[Sending a WebSphere MQ endpoint request](#)

[Testing all operations in a WSDL file](#)

[Synchronizing a remote WSDL file](#)

[Adding static XML headers to a service request](#)

[Opening file attachments](#)

Related information:

[Creating transport protocol configurations](#)

Synchronizing a remote WSDL file

For web services that make their Web Services Description Language (WSDL) file available from a URL, you might have to ensure that the WSDL that you work with is always up to date. By synchronizing the WSDL, you ensure that the local copy of the WSDL in your workspace is regularly synchronized with the remote WSDL.

Before you begin

Ensure that you have a valid WSDL file. Ensure that the WSDLs use the correct syntax for the test environment. The product might not work with some WSDL files. WSDL synchronization only works with remote WSDLs that are imported from a URL.

The WSDL synchronization runs either when the workbench is started or after a specified period. If the remote WSDL changes, the local copy of the WSDL is updated. Depending on the changes, a merge is performed and any service requests that use the WSDL are updated. If the changes to the WSDL cannot be automatically applied to the service requests, for example if an operation is removed or renamed or if the XML structure of the service request is changed, the test is marked with a error.

Procedure

To import a synchronized remote WSDL:

1. Open the generic service client, click the **Requests** tab, and then, click **Add a WSDL file**.
2. In the Add WSDL Files window, click **Import from URL** to download and import a remote WSDL from the web.
3. On the Import WSDL from URL page, type the URL of the remote WSDL. If you are connecting through a proxy or a corporate firewall, click **Proxy properties** to specify your network settings.
4. In the **Synchronization policy** area, specify whether and when to synchronize WSDLs:
 - Select **Never** if you do not want the remote WSDL to be updated.
 - Select **On session launch** to synchronize the WSDL each time you start the workbench.
 - Select **Every** to specify a synchronization period in days.
5. Click **OK**. The WSDL is added to the Call Library.

What to do next

After the WSDL is imported, you can change the synchronization settings by right-clicking the WSDL in the generic service client Call Library or in the test navigator. Then select **WSDL Synchronization**. The WSDL Synchronization window also displays the date of the latest synchronization.

Related concepts:

[Generic service client overview](#)

Related tasks:

- [Sending service requests with WSDL files](#)
- [Sending HTTP endpoint requests](#)
- [Sending a JMS endpoint request](#)
- [Sending a WebSphere MQ endpoint request](#)
- [Testing all operations in a WSDL file](#)
- [Viewing message content](#)
- [Adding static XML headers to a service request](#)
- [Opening file attachments](#)
- [Sending service requests with WSDL files](#)
- [Testing all operations in a WSDL file](#)

Related information:

- [Creating transport protocol configurations](#)

Adding static XML headers to a service request

You can add static XML headers to service requests to ensure compliance with WS-Addressing, WS-ReliableMessaging, and WS-Coordination specifications as well as other predefined standards.

About this task

Static XML headers are compliant with the web service specifications for service-oriented architecture (SOA). Checks are performed to ensure that the XML headers are valid.

Procedure

To add a static XML header to a request:

1. Open a service request in the generic service client. The location of the XML header depends on the product that you are using:
 - For IBM® Security AppScan®, click the Request Stack tab and in the algorithm stack for the request, click **Add > Static XML Headers**.
 - For IBM Rational® Performance Tester, IBM Rational Service Tester for SOA Quality, or other products, click the Message tab and click **Form**.
2. On the **Header** bar, click **Add (+)** to open the menu.
3. Select the web service specification for the request to be comply with, or click **More** to open a detailed list of specifications. The XML structure of the header is created.
4. Edit the header as required. Some elements require completion or content to be specified. XML elements that are invalid or require attention are marked with a warning or an error symbol.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Sending service requests with WSDL files](#)
[Sending HTTP endpoint requests](#)
[Sending a JMS endpoint request](#)
[Sending a WebSphere MQ endpoint request](#)
[Testing all operations in a WSDL file](#)
[Viewing message content](#)
[Synchronizing a remote WSDL file](#)
[Opening file attachments](#)
[Editing WSDL security profiles](#)
[Adding WS-Addressing to a security configuration](#)

Related information:

[Creating transport protocol configurations](#)

Opening file attachments

When a service sends a file attachment with the response, you must import it as a resource to open the attachment.

Before you begin

Ensure that you have specified an editor to view the attachment type in. Click **Window > Preferences > General > Editors > File associations**, and specify the editor.

Procedure

1. Open the message return, and click the **Attachment** tab. File attachments are listed with a default name, a MIME type, and a contents ID.
2. Select the line for the attachment that you want to open, and click **Open**.
3. In the Create Resource window, type a name for the resource, and select a location where it will be imported, and click **OK**. Ensure that the name of the resource includes a file extension that is compatible with the MIME type of the attachment.

What to do next

After the attachment has been imported, you can click on **Open** again to open the file in the corresponding editor.

Related concepts:

[Generic service client overview](#)

Related tasks:

[Sending service requests with WSDL files](#)
[Sending HTTP endpoint requests](#)
[Sending a JMS endpoint request](#)
[Sending a WebSphere MQ endpoint request](#)
[Testing all operations in a WSDL file](#)
[Viewing message content](#)
[Synchronizing a remote WSDL file](#)
[Adding static XML headers to a service request](#)

Related information:

[Creating transport protocol configurations](#)

Generic service client reference

- **Generic service client call details**

In the generic service client, service calls contain the content and the transport information for the call. The contents are made of the SOAP envelope. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

- **Generic service client binary call details**

In the generic service client, binary calls are specialized calls for sending binary messages. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

- **Generic service client binary call details**

In the generic service client, binary calls are specialized calls for sending binary messages. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

- **Generic service client message return details**

In the generic service client, message returns are generated after a service call is successfully invoked. Message returns display the content returned by the service.

Related reference:

[WSDL security editor reference](#)

Generic service client call details

In the generic service client, service calls contain the content and the transport information for the call. The contents are made of the SOAP envelope. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

Message

This page shows the XML content of the request and provides access to data correlation. The same content is presented in three different ways.

- Form

- This view provides a simplified view of the message that focuses on editing the values of the XML content. Use the **Schema** menu to enable assistance with editing XML content so that the XML is valid and complies with the XSD specification. In the Form view, add the XML headers that are required for standard web service calls. On the **Header** bar, click **Add (+)** to create the default XML header structure for WS-Addressing, WS-ReliableMessaging or WS-Coordination requests, or click **More** for other standards. You can enable or disable XML header elements and specify the correct values for each XML element. Checks are performed to ensure that the XML content is valid.

Note: To add XML headers to calls in IBM® Security AppScan®, add a **Static XML Headers** algorithm on the Request Stack tab of the request.

- Tree

- This view provides a hierarchical view of the XML structure of the message, including elements, namespaces, and the associated values. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit the XML elements and namespaces in the tree.

Use **Skip if Empty** column to select the empty XML elements that you want to skip. This column is visible only if you selected the **Display the 'Skip if Empty' column in XML tree viewer** check box in **Window > Preferences > Test > Test editor > Service test**.

Click **Filter** to hide or show namespace, attribute, or text nodes, depending on your requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable **Allow only valid modifications** if you do not have an XSD or if you want to bypass the schema.

You can right-click an XML element to convert it to an XML fragment. This enables you to perform data correlation (use datapools and create references) on the entire XML fragment instead of only on the value.

- Source

- This view displays the source XML content of the message or plain text content. To format XML content, click **Format XML text**. To wrap XML content into a single line, click **Pack XML text to single line**. Similar controls are available for JSON content. **Important:** In the Source view, do not edit the tags that start with `SoaTag`. If you delete or change these tags, any references and substitutions in the test will be broken. You cannot recreate these tags after you delete them.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments. The method for using this identifier depends on your server requirements.

- MIME or DIME

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- Use MTOM transmission mechanism

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Transport

This page covers the transport settings used to send the request. The transport protocol settings apply to a transport configuration, which can be either HTTP, Java™ Message Service (JMS), WebSphere® MQ, or Microsoft .NET. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM Security AppScan, only the HTTP transport protocol is available.

- HTTP

- Select **HTTP** to use the HTTP transport for the request. At the request level, you can update a URL or SOAP action and the reference to the global configuration of a test.
 - **Protocol configuration**
 - Click **Change** to specify a predefined transport configuration or to create a configuration. HTTP transport configurations contain proxy and authentication settings that can be reused.
 - **URL**
 - Specify the URL end point of the service request.
 - **Rest mode**
 - Use this check box to split the REST URL so that it is easy to understand the different parts of REST URL. When you use this option, the main section of URL is placed in the URL field, the resource part is placed in the **Resource** field, and the parameters are placed in the **Parameters** field. Use the **Add** button to manually add more parameters.
 - **Method and Version**
 - Specify the HTTP method and version to be used to invoke the service request.
 - **Headers**
 - Specify the names and values of any custom HTTP headers that are required by the service. Click **Add**, **Edit** or **Remove** to modify the headers

list.

- **Cookies**

- Specify the names and values of any cookies that are required by the service. Click **Add**, **Edit** or **Remove** to modify the cookies list.

- **JMS**

- Select **JMS** to use the Java Messaging Service transport for the request. This page enables you to add string properties that are attached to the request for a JMS configuration. These will be sent as message properties through JMS.

- **Protocol configuration**

- Click **Change** to specify a predefined transport configuration or to create a configuration. JMS transport configurations contain generic end point, reception point, and adapter settings that can be reused.

- **Properties**

- Specify the names and values of any string properties that are required by the request for the current JMS transport configuration. These are sent as message properties through JMS. Click **Add**, **Edit** or **Remove** to modify the properties list.

- **WebSphere MQ**

- Select **MQ** to use the IBM WebSphere MQ transport for the request. This page enables you to specify the SOAP action and override the settings for the WebSphere MQ configuration selected at the test level.

- **Protocol configuration**

- Click **Change** to specify a predefined transport configuration or to create a configuration. WebSphere MQ transport configurations contain generic queue, header, and SSL settings that can be reused.

- **SOAP Action**

- Specifies the SOAP action to be used to invoke the WebSphere MQ request.

- **Override MQ protocol configuration values**

- Select this option to configure the fields of the WebSphere MQ message. You can replace a subset of an MQ message descriptor with a custom format for use with other server types, specifically when using an XML message request.

- **Customize message header**

- Select this option to specify custom headers for the transport for the SOAP over MQ feature that is provided by WebSphere MQ. This feature uses a predetermined MQ message format (RFH2), therefore, when selected, other **Message Descriptor** options are disabled.

- **Message descriptor**

- These settings replace the message descriptor and header settings of the MQ protocol configuration. Refer to WebSphere MQ documentation for information about message descriptors.

- **Microsoft .NET**

- Select **Microsoft .NET** to use the Microsoft .NET Framework transport for requests based on Windows Communication Foundation (WCF). This page enables you to override the settings for the Microsoft .NET configuration selected at the test level.

- **Item**

- Click **Add** to specify the name and value of the WCF actions that are required by the service. This table is automatically generated when you import a Microsoft .NET WSDL file. Refer to the Microsoft .NET WCF documentation for more information.

Request Stack

Use this page to specify the stack that applies security and addressing parameters and algorithms to service requests before they are sent. Stacks are a set of algorithms that are executed in a given order. Use the WSDL security editor to define a stack for each WSDL. The stack will be applied to all requests that use the WSDL.

- **Override stack**

- By default, you edit a stack which attached to a specific WSDL file in the WSDL security editor. Select this option to specify a different security algorithm stack only for the current service request.

- **Show response stack**

- The Request Stack page contains algorithms that are applied only to outgoing service requests. Select **Show response stack** to add a Response Stack page. The Response Stack page allows you to edit security and addressing parameters and algorithms that are applied to incoming responses.

- **Security Algorithm Details**

- Click **Add**, **Insert**, or **Remove** to add or remove security algorithms in the stack. Click **Up** and **Down** to change the order of a selected algorithm in the security stack. The following security algorithms can be added to the security stack:

- **Static XML Headers**

- Use this algorithm to add the XML headers that are required for web service standard calls. On the **Header** bar, click **Add (+)** to create the default XML header structure for WS-Addressing, WS-ReliableMessaging or WS-Coordination requests, or click **More** for other standards. You can enable or disable XML elements in the Header section and specify the correct values for each XML element. Checks are performed to ensure that the XML headers are valid.

Note: The Static XML Headers algorithm is available only in IBM Security AppScan. To add static XML headers to calls in other products, expand the **Headers** section on the Message tab of the request.

- **Time Stamp**

- The time stamp security algorithm adds time stamp information to the XML document in the response. For details on security algorithms, refer to the web service security specification.

-

Actor / Role name

- Specify the name of the recipient of the algorithm header element, if required.

-

Must understand

- Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or

the server.

- Expiration delay

- Specify the delay after which the time stamp expires.

- Millisecond precision

- Select this option to produce a time stamp that uses millisecond precision instead of the default (1/100th second).

- User name token

- The user name token security algorithm adds a user name token to the XML document in the message. For details on security algorithms, refer to the web service security specification.

- Actor / Role name

- Specify the name of the recipient of the algorithm header element, if required.

- Must understand

- Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.

- Name

- Type the name of the user.

- Password

- Type the password of the user.

- Password type

- Specify the password type for the security algorithm as defined in the Web Services Security UsernameToken profile.

- XML Encryption

- The XML encryption security algorithm specifies how the XML document is encrypted. For details on security algorithms, refer to the web service security specification.

- Actor / Role name

- Specify the name of the recipient of the algorithm header element, if required.

- Must understand

- Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.

- Identifier type

- Select the type of key identifier to be used for the encryption. The following key identifiers are available, as defined in the Web Services Security (WSS) specification X509 profile and the OASIS WSS 1.1 specification:

- ISSUER_SERIAL

- BST_DIRECT_REFERENCE

- X509_KEY_IDENTIFIER

- SKI_KEY_IDENTIFIER

- EMBEDDED_KEYNAME

- THUMBPRINT_IDENTIFIER

- ENCRYPTED_KEY_SHA1_IDENTIFIER

- **User XPath part selection**
 - This enables you to specify an XPath query that describes parts of the XML document that can be subjects of the algorithm. By default, the body is the subject.
 - **Key**
 - Select the key used for the encryption. The details of each key vary.
 - **x509 key**: This specifies the name and password of the x509 key and the keystore where it is located.
 - **Raw key**: This specifies the name and the byte value of your SecretKey in hexadecimal.
 - **Encrypted key**: This specifies a reference to an encrypted key that was previously defined in the security stack. Click **Insert a new encrypted key** to create a new encrypted key definition block.
 - **Encoding Algorithm Name**
 - Specify the encryption method to be used as defined in the XML Encryption Syntax and Processing specification.
 - **Key Encoding Algorithm**
 - Specify the standard algorithm for encoding the key as defined in the XML Encryption Syntax and Processing specification.
 - **XML Signature**
 - The XML signature security algorithm specifies how the XML document is signed. For details on security algorithms, refer to the web service security specification.
 - **Actor / Role name**
 - Specify the name of the recipient of the algorithm header element, if required.
 - **Must understand**
 - Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.
 - **Security token**
 - Select the type of key identifier to be used for the signature. The following key identifiers are available, as defined in the the Web Service Security (WSS) specification X509 profile and OASIS WSS 1.1 specification:
 - ISSUER_SERIAL
 - BST_DIRECT_REFERENCE
 - X509_KEY_IDENTIFIER
 - SKI_KEY_IDENTIFIER
 - KEY_VALUE
 - USER_NAME_TOKEN
 - CUSTOM_SYMM_SIGNATURE
- In addition, the following identifiers are available when the signature is based on a UsernameToken profile:
- USER_NAME_TOKEN
 - CUSTOM_SYMM_SIGNATURE
-

- **User XPath part selection**
 - Specify an XPath query that describes parts of the XML document that can be subjects of the algorithm. By default, the body is the subject. Click the **XPath Helper** button to build the Xpath expression.
- **Key**
 - Select the key used for the encryption. The details of each key vary.
 - **x509 key**: This specifies the name and password of the x509 key and the keystore where it is located.
 - **User name token key**: This specifies a user name and password for the signature.
 - **Encrypted key**: This specifies a reference to an encrypted key that was previously defined in the security stack. Click **Insert a new encrypted key** to create a new encrypted key definition block.
- **Signature algorithm name**
 - Specify the signature method algorithm as described in the XML Signature Syntax and Processing specification.
- **Canonicalization**
 - Specify the canonicalization method to be used as described in the XML Signature Syntax and Processing specification.
- **Inclusive namespaces**
 - Specify whether the canonicalization is exclusive as described in the Exclusive XML Canonicalization specification.
- **Encrypted Key**
 - This block defines an encrypted key that can be used in an XML signature or XML encryption block. The encrypted key block must be before a block that uses the encrypted key.
 - **Actor / Role name**
 - Specify the name of the recipient of the algorithm header element, if required.
 - **Must understand**
 - Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.
 - **Key name**
 - Specify the name of the encrypted key.
 - **Identifier type**
 - Select the type of key identifier to be used for the key. The following key identifiers are available, as defined in the the Web Service Security (WSS) specification X509 profile and OASIS WSS 1.1 specification:
 - ISSUER_SERIAL
 - BST_DIRECT_REFERENCE
 - X509_KEY_IDENTIFIER
 - THUMBPRINT_IDENTIFIER
 - SKI_KEY_IDENTIFIER
 - **Key size**

- Specify the size of the key in bits.
- **Key encoding algorithm name**
 - Specify the algorithm to be used for encoding the key.
- **Keystore**
 - Select a keystore or click **Edit Security** to define a new keystore or to manage the existing keystores.
- **Name**
 - Select a key contained in the specified keystore.
- **Password**
 - Type the password for the selected key name.
- **Custom Security Algorithm**
 - If you want to use a Java class as a custom security algorithm, then use this stack element to apply the custom algorithm to the service.
 - **Java Project**
 - If you have not implemented a custom Java class, select **Java Project**, type a name for the new project, and click **Generate** to create a new Java class with the default structure for custom security implementations. **Note:** If you are using IBM Security AppScan, this field is not available.
 - **Implementation class**
 - Specify the name of the class that implements the custom security algorithm. Click **Browse Class** to select an existing Java class from the workspace.
 - **Properties**
 - Use this table to send any specific properties and associated values to the custom security algorithm.
- **WS-Addressing Algorithm**
 - Use this block if your service uses either WS-Addressing 2004/08 or the WS-Addressing 1.0 Core standard.
 - **Namespace**
 - Specify the namespace for either WS-Addressing 2004/08 or WS-Addressing 1.0 Core.
 - **Action if request uses WS-Addressing**
 - Select the action to complete if WS-Addressing is already in the request.
 - **Replace anonymous address in Reply-to with:**
 - Select this option to generate the specified address in the Reply-to header instead of an anonymous address.
 - **Remove WS-Addressing from response**
 - Select this option to strip any WS-Addressing headers from the response.
- **WS-Policy Algorithm**
 - Use this block if your service requires a security policy file compliant with the WS-Policy specification.
 - **Use policy included in WSDL (WS-PolicyAttachment)**
 - Select this option to use the security policy configuration that is attached to the WSDL as in the WS-PolicyAttachment specification.

- **Policy**

- If you are not using the WS-PolicyAttachment specification, specify the XML policy file. Click **Browse** to add a policy file from the workspace or to import a policy file.

- **Signature configuration**

- Select this option to specify a keystore for any signature that is specified in the policy. Click **Edit Security** to add a keystore from the workspace or to import a keystore.

- **Encryption configuration**

- Select this option to specify a keystore for any encryption that is specified in the policy. Click **Edit Security** to add a keystore from the workspace or to import a keystore.

- **Decryption configuration**

- Select this option to specify a keystore for any decryption that is specified in the policy. Click **Edit Security** to add a keystore from the workspace or to import a keystore.

- **Retrieve token from security token server (WS-Trust)**

- Select this option, and click **Configure** to specify a Security Token Server (STS) to use with the policy.

- **Additional properties**

- Use this table to specify settings for the advanced properties or specific implementations of the WS-Security specification. Click **Add** to add a property name and to set a value.

Response Stack

Use this page to specify the stack that applies security and addressing parameters to responses after they are received. Stacks are a set of algorithms that are executed in a given order. Use the WSDL security editor to define a stack for each WSDL. The stack will be applied to all requests that use the WSDL.

- **Override stack**

- By default, you edit the security algorithm stack attached to a specific WSDL file in the WSDL Security Editor. Select this option to specify a different security algorithm stack only for the current response.

- **Show response stack**

- Clear the **Show response stack** option to hide the Response Stack page.

- **Security Algorithm Details**

- Click **Add**, **Insert**, or **Remove** to add or remove security algorithms in the stack. Click **Up** and **Down** to change the order of a selected algorithm in the security stack. The following security algorithms can be added to the security stack:

- **XML Encryption**

- The XML encryption security algorithm specifies how the XML document is encrypted. For details on security algorithms, refer to the web service security specification.

- **Actor / Role name**

- Specify the name of the recipient of the algorithm header element, if required.

- **Must understand**
 - Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.
- **Identifier type**
 - Select the type of key identifier to be used for the encryption. The following key identifiers are available, as defined in the Web Services Security (WSS) specification X509 profile and the OASIS WSS 1.1 specification:
 - ISSUER_SERIAL
 - BST_DIRECT_REFERENCE
 - X509_KEY_IDENTIFIER
 - SKI_KEY_IDENTIFIER
 - EMBEDDED_KEYNAME
 - THUMBPRINT_IDENTIFIER
 - ENCRYPTED_KEY_SHA1_IDENTIFIER
- **User XPath part selection**
 - This enables you to specify an XPath query that describes parts of the XML document that can be subjects of the algorithm. By default, the body is the subject.
- **Key**
 - Select the key used for the encryption. The details of each key vary.
 - **x509 key**: This specifies the name and password of the x509 key and the keystore where it is located.
 - **Raw key**: This specifies the name and the byte value of your SecretKey in hexadecimal.
 - **Encrypted key**: This specifies a reference to an encrypted key that was previously defined in the security stack. Click **Insert a new encrypted key** to create a new encrypted key definition block.
- **Encoding Algorithm Name**
 - Specify the encryption method to be used as defined in the XML Encryption Syntax and Processing specification.
- **Key Encoding Algorithm**
 - Specify the standard algorithm for encoding the key as defined in the XML Encryption Syntax and Processing specification.
- **Encrypted Key**
 - This block defines an encrypted key that can be used in an XML signature or XML encryption block. The encrypted key block must be before a block that uses the encrypted key.
 - **Actor / Role name**
 - Specify the name of the recipient of the algorithm header element, if required.
 - **Must understand**
 - Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.
 - **Key name**

- Specify the name of the encrypted key.
- **Identifier type**
 - Select the type of key identifier to be used for the key. The following key identifiers are available, as defined in the the Web Service Security (WSS) specification X509 profile and OASIS WSS 1.1 specification:
 - ISSUER_SERIAL
 - BST_DIRECT_REFERENCE
 - X509_KEY_IDENTIFIER
 - THUMBPRINT_IDENTIFIER
 - SKI_KEY_IDENTIFIER
- **Key size**
 - Specify the size of the key in bits.
- **Key encoding algorithm name**
 - Specify the algorithm to be used for encoding the key.
- **Keystore**
 - Select a keystore or click **Edit Security** to define a new keystore or to manage the existing keystores.
- **Name**
 - Select a key contained in the specified keystore.
- **Password**
 - Type the password for the selected key name.
- **Custom Security Algorithm**
 - If you want to use a Java class as a custom security algorithm, then use this stack element to apply the custom algorithm to the service.
 - **Java Project**
 - If you have not implemented a custom Java class, select **Java Project** , type a name for the new project, and click **Generate** to create a new Java class with the default structure for custom security implementations.**Note:** If you are using IBM Security AppScan, this field is not available.
 - **Implementation class**
 - Specify the name of the class that implements the custom security algorithm. Click **Browse Class** to select an existing Java class from the workspace.
 - **Properties**
 - Use this table to send any specific properties and associated values to the custom security algorithm.

Generic service client binary call details

In the generic service client, binary calls are specialized calls for sending binary messages. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

Message

- Update node name automatically

- Select this option to automatically rename the request in the Test Contents view.

- Do not wait for response

- Select this option to skip directly to the next request in the test after the current request is sent.

- Time Out (ms)

- This is the timeout value in milliseconds. If no response is received after the specified time, an error is produced.

- Think Time (ms)

- This specifies the programmatically calculated time delay that is observed for each user when this test is run with multiple virtual users. Think time is a statistical emulation of the amount of time actual users spend reading or thinking before performing an action.

- Update Response

- Click this button to invoke the request with the current settings and to use the response to create a binary response element or to update the existing response element.

- Source

- This page presents the binary contents of the request and provides access to data correlation. The same contents are presented in Binary and Raw ASCII views.

- Attachments

- This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties.

- Transport

- This page covers the transport protocol used to send the request. The transport protocol can be either HTTP, Java™ Message Service (JMS), or WebSphere® MQ. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM® Security AppScan®, only the HTTP transport protocol is available.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments.

The method for using this identifier depends on your server requirements.

- MIME or DIME

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- Use MTOM transmission mechanism

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Transport

This page covers the transport settings used to send the request. The transport protocol settings apply to a transport configuration, which can be either HTTP, Java Message Service (JMS), WebSphere MQ, or Microsoft .NET. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM Security AppScan, only the HTTP transport protocol is available.

- HTTP

- Select **HTTP** to use the HTTP transport for the request. At the request level, you can update a URL or SOAP action and the reference to the global configuration of a test.

- Protocol configuration

- Click **Change** to specify a predefined transport configuration or to create a configuration. HTTP transport configurations contain proxy and authentication settings that can be reused.

- URL

- Specify the URL end point of the service request.

- Rest mode

- Use this check box to split the REST URL so that it is easy to understand the different parts of REST URL. When you use this option, the main section of URL is placed in the URL field, the resource part is placed in the **Resource** field, and the parameters are placed in the **Parameters** field. Use the **Add** button to manually add more parameters.

- Method and Version

- Specify the HTTP method and version to be used to invoke the service request.

- Headers

- Specify the names and values of any custom HTTP headers that are required by the service. Click **Add**, **Edit** or **Remove** to modify the headers list.

- Cookies

- Specify the names and values of any cookies that are required by the service. Click **Add**, **Edit** or **Remove** to modify the cookies list.

- JMS

- Select **JMS** to use the Java Messaging Service transport for the request. This page enables you to add string properties that are attached to the request for a

JMS configuration. These will be sent as message properties through JMS.

- Protocol configuration

- Click **Change** to specify a predefined transport configuration or to create a configuration. JMS transport configurations contain generic end point, reception point, and adapter settings that can be reused.

- Properties

- Specify the names and values of any string properties that are required by the request for the current JMS transport configuration. These are sent as message properties through JMS. Click **Add**, **Edit** or **Remove** to modify the properties list.

- WebSphere MQ

- Select **MQ** to use the IBM WebSphere MQ transport for the request. This page enables you to specify the SOAP action and override the settings for the WebSphere MQ configuration selected at the test level.

- Protocol configuration

- Click **Change** to specify a predefined transport configuration or to create a configuration. WebSphere MQ transport configurations contain generic queue, header, and SSL settings that can be reused.

- SOAP Action

- Specifies the SOAP action to be used to invoke the WebSphere MQ request.

- Override MQ protocol configuration values

- Select this option to configure the fields of the WebSphere MQ message. You can replace a subset of an MQ message descriptor with a custom format for use with other server types, specifically when using an XML message request.

- Customize message header

- Select this option to specify custom headers for the transport for the SOAP over MQ feature that is provided by WebSphere MQ. This feature uses a predetermined MQ message format (RFH2), therefore, when selected, other **Message Descriptor** options are disabled.

- Message descriptor

- These settings replace the message descriptor and header settings of the MQ protocol configuration. Refer to WebSphere MQ documentation for information about message descriptors.

- Microsoft .NET

- Select **Microsoft .NET** to use the Microsoft .NET Framework transport for requests based on Windows Communication Foundation (WCF). This page enables you to override the settings for the Microsoft .NET configuration selected at the test level.

- Item

- Click **Add** to specify the name and value of the WCF actions that are required by the service. This table is automatically generated when you import a Microsoft .NET WSDL file. Refer to the Microsoft .NET WCF documentation for more information.

Generic service client text call details

In the generic service client, text calls are specialized calls for sending text messages. The transport information refers to the information that is required to send and receive and answer depending on the selected protocol.

Message

- Update node name automatically

- Select this option to automatically rename the request in the Test Contents view.

- Do not wait for response

- Select this option to skip directly to the next request in the test after the current request is sent.

- Time Out (ms)

- This is the timeout value in milliseconds. If no response is received after the specified time, an error is produced.

- Think Time (ms)

- This specifies the programmatically calculated time delay that is observed for each user when this test is run with multiple virtual users. Think time is a statistical emulation of the amount of time actual users spend reading or thinking before performing an action.

- Update Response

- Click this button to invoke the request with the current settings and to use the response to create a binary response element or to update the existing response element.

- Source

- This page presents the binary contents of the request and provides access to data correlation. The same contents are presented in Binary and Raw ASCII views.

- Attachments

- This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties.

- Transport

- This page covers the transport protocol used to send the request. The transport protocol can be either HTTP, Java™ Message Service (JMS), or WebSphere® MQ. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM® Security AppScan®, only the HTTP transport protocol is available.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments.

The method for using this identifier depends on your server requirements.

- **MIME or DIME**

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- **Use MTOM transmission mechanism**

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Transport

This page covers the transport settings used to send the request. The transport protocol settings apply to a transport configuration, which can be either HTTP, Java Message Service (JMS), WebSphere MQ, or Microsoft .NET. You can create several configurations for each protocol so that you can easily switch protocols or variants of protocols. **Note:** If you are using IBM Security AppScan, only the HTTP transport protocol is available.

- **HTTP**

- Select **HTTP** to use the HTTP transport for the request. At the request level, you can update a URL or SOAP action and the reference to the global configuration of a test.

- **Protocol configuration**

- Click **Change** to specify a predefined transport configuration or to create a configuration. HTTP transport configurations contain proxy and authentication settings that can be reused.

- **URL**

- Specify the URL end point of the service request.

- **Rest mode**

- Use this check box to split the REST URL so that it is easy to understand the different parts of REST URL. When you use this option, the main section of URL is placed in the URL field, the resource part is placed in the **Resource** field, and the parameters are placed in the **Parameters** field. Use the **Add** button to manually add more parameters.

- **Method and Version**

- Specify the HTTP method and version to be used to invoke the service request.

- **Headers**

- Specify the names and values of any custom HTTP headers that are required by the service. Click **Add**, **Edit** or **Remove** to modify the headers list.

- **Cookies**

- Specify the names and values of any cookies that are required by the service. Click **Add**, **Edit** or **Remove** to modify the cookies list.

- **JMS**

- Select **JMS** to use the Java Messaging Service transport for the request. This page enables you to add string properties that are attached to the request for a

JMS configuration. These will be sent as message properties through JMS.

- Protocol configuration

- Click **Change** to specify a predefined transport configuration or to create a configuration. JMS transport configurations contain generic end point, reception point, and adapter settings that can be reused.

- Properties

- Specify the names and values of any string properties that are required by the request for the current JMS transport configuration. These are sent as message properties through JMS. Click **Add**, **Edit** or **Remove** to modify the properties list.

- WebSphere MQ

- Select **MQ** to use the IBM WebSphere MQ transport for the request. This page enables you to specify the SOAP action and override the settings for the WebSphere MQ configuration selected at the test level.

- Protocol configuration

- Click **Change** to specify a predefined transport configuration or to create a configuration. WebSphere MQ transport configurations contain generic queue, header, and SSL settings that can be reused.

- SOAP Action

- Specifies the SOAP action to be used to invoke the WebSphere MQ request.

- Override MQ protocol configuration values

- Select this option to configure the fields of the WebSphere MQ message. You can replace a subset of an MQ message descriptor with a custom format for use with other server types, specifically when using an XML message request.

- Customize message header

- Select this option to specify custom headers for the transport for the SOAP over MQ feature that is provided by WebSphere MQ. This feature uses a predetermined MQ message format (RFH2), therefore, when selected, other **Message Descriptor** options are disabled.

- Message descriptor

- These settings replace the message descriptor and header settings of the MQ protocol configuration. Refer to WebSphere MQ documentation for information about message descriptors.

- Microsoft .NET

- Select **Microsoft .NET** to use the Microsoft .NET Framework transport for requests based on Windows Communication Foundation (WCF). This page enables you to override the settings for the Microsoft .NET configuration selected at the test level.

- Item

- Click **Add** to specify the name and value of the WCF actions that are required by the service. This table is automatically generated when you import a Microsoft .NET WSDL file. Refer to the Microsoft .NET WCF documentation for more information.

Generic service client message return details

In the generic service client, message returns are generated after a service call is successfully invoked. Message returns display the content returned by the service.

Message

This page shows the XML content of the request and provides access to data correlation. The same content is presented in three different ways.

- Form

- This view provides a simplified view of the message that focuses on editing the values of the XML content. Use the **Schema** menu to enable assistance with editing XML content so that the XML is valid and complies with the XSD specification. In the Form view, add the XML headers that are required for standard web service calls. On the **Header** bar, click **Add (+)** to create the default XML header structure for WS-Addressing, WS-ReliableMessaging or WS-Coordination requests, or click **More** for other standards. You can enable or disable XML header elements and specify the correct values for each XML element. Checks are performed to ensure that the XML content is valid.

Note: To add XML headers to calls in IBM® Security AppScan®, add a **Static XML Headers** algorithm on the Request Stack tab of the request.

- Tree

- This view provides a hierarchical view of the XML structure of the message, including elements, namespaces, and the associated values. You can use **Add**, **Insert**, **Remove**, **Up**, and **Down** to edit the XML elements and namespaces in the tree.

Use **Skip if Empty** column to select the empty XML elements that you want to skip. This column is visible only if you selected the **Display the 'Skip if Empty' column in XML tree viewer** check box in **Window > Preferences > Test > Test editor > Service test**.

Click **Filter** to hide or show namespace, attribute, or text nodes, depending on your requirements.

Click **Allow only valid modifications** to enable smart editing, based on a specified XML schema document (XSD). To specify a set of XSD documents for the workbench, in the test navigator, right-click the project and select **Properties** and **Schema Catalog**. Disable **Allow only valid modifications** if you do not have an XSD or if you want to bypass the schema.

You can right-click an XML element to convert it to an XML fragment. This enables you to perform data correlation (use datapools and create references) on the entire XML fragment instead of only on the value.

- Source

- This view displays the source XML content of the message or plain text content. To format XML content, click **Format XML text**. To wrap XML content into a single line, click **Pack XML text to single line**. Similar controls are available for JSON content. **Important:** In the Source view, do not edit the tags that start with `SoaTag`. If you delete or change these tags, any references and substitutions in the test will be broken. You cannot recreate these tags after you delete them.

Attachments

This page lists the MIME or DIME attachments that are attached to the request. The contents of this view conform to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification. You can use this page to add workbench resources as MIME or DIME attachments and change properties. The **Content ID** is the identifier that the request uses to refer to the attachments. The method for using this identifier depends on your server requirements.

- MIME or DIME

- Select whether the attachment conforms to the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) specification

- Use MTOM transmission mechanism

- By default, the request uses SOAP Messages with Attachments (SwA) to handle attachments. Select this option to handle attachments with the SOAP Message Transmission Optimization Mechanism (MTOM).

Response Properties

This page lists the names and values of properties of the response.

WSDL security editor reference

With the Web Service Description Language (WSDL) security editor you can create and edit security configurations for a WSDL file.

Keystores

In this page, you can edit the keystores that are used for the WSDL file. The keystore contains the public and private keys that are required for the specified security protocol.

- Defined Keystores

- Click **Add** or **Remove** to add or remove keystore files from the workbench.

- Keystore Details

- This specifies the location and file name of the selected keystore. Click **Browse** to select a different file.

- Name

- This specifies the name of the keystore. This name is used throughout the test instead of the file name.

- File

- Click **Browse** to specify a keystore file containing a valid server certificate. The following formats are supported:

- KS
- JKS
- JCEKS
- PKCS12 (p12 or PFX)
- PEM

- Password

- If the keystore file is encrypted, type the required password.

Security Stacks

In this page you can edit the security algorithm stacks that the security protocol uses. Security stacks are a set of algorithms that are executed in a given order.

- Security Stacks

- Click **Add**, **Remove**, or **Rename** to add, remove, or rename the security stacks that are associated with the WSDL file.

- Security Algorithm Details

- Click **Add**, **Insert**, or **Remove** to add or remove security algorithms in the stack. Click **Up** and **Down** to change the order of a selected algorithm in the security stack. The following security algorithms can be added to the security stack:

- Time Stamp

- The time stamp security algorithm adds time stamp information to the XML document in the response. For details on security algorithms, refer to the web service security specification.

-

Actor / Role name

- Specify the name of the recipient of the algorithm header element, if required.

- - Must understand**
 - Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.
 - **Expiration delay**
 - Specify the delay after which the time stamp expires.
 - **Millisecond precision**
 - Select this option to produce a time stamp that uses millisecond precision instead of the default (1/100th second).
- **User name token**
 - The user name token security algorithm adds a user name token to the XML document in the message. For details on security algorithms, refer to the web service security specification.
 - **Actor / Role name**
 - Specify the name of the recipient of the algorithm header element, if required.
 - **Must understand**
 - Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.
 - **Name**
 - Type the name of the user.
 - **Password**
 - Type the password of the user.
 - **Password type**
 - Specify the password type for the security algorithm as defined in the Web Services Security UsernameToken profile.
- **XML Encryption**
 - The XML encryption security algorithm specifies how the XML document is encrypted. For details on security algorithms, refer to the web service security specification.
 - **Actor / Role name**
 - Specify the name of the recipient of the algorithm header element, if required.
 - **Must understand**
 - Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.
 - **Identifier type**
 - Select the type of key identifier to be used for the encryption. The following key identifiers are available, as defined in the Web Services Security (WSS) specification X509 profile and the OASIS WSS 1.1 specification:
 - ISSUER_SERIAL
 - BST_DIRECT_REFERENCE
 - X509_KEY_IDENTIFIER

- SKI_KEY_IDENTIFIER
- EMBEDDED_KEYNAME
- THUMBPRINT_IDENTIFIER
- ENCRYPTED_KEY_SHA1_IDENTIFIER
- **User XPath part selection**
 - This enables you to specify an XPath query that describes parts of the XML document that can be subjects of the algorithm. By default, the body is the subject.
- **Key**
 - Select the key used for the encryption. The details of each key vary.
 - **x509 key**: This specifies the name and password of the x509 key and the keystore where it is located.
 - **Raw key**: This specifies the name and the byte value of your SecretKey in hexadecimal.
 - **Encrypted key**: This specifies a reference to an encrypted key that was previously defined in the security stack. Click **Insert a new encrypted key** to create a new encrypted key definition block.
- **Encoding Algorithm Name**
 - Specify the encryption method to be used as defined in the XML Encryption Syntax and Processing specification.
- **Key Encoding Algorithm**
 - Specify the standard algorithm for encoding the key as defined in the XML Encryption Syntax and Processing specification.
- **XML Signature**
 - The XML signature security algorithm specifies how the XML document is signed. For details on security algorithms, refer to the web service security specification.
 - **Actor / Role name**
 - Specify the name of the recipient of the algorithm header element, if required.
 - **Must understand**
 - Select whether it is mandatory that the algorithm header is processed by the recipient, if required. The recipient is either the Actor name or the server.
 - **Security token**
 - Select the type of key identifier to be used for the signature. The following key identifiers are available, as defined in the the Web Service Security (WSS) specification X509 profile and OASIS WSS 1.1 specification:
 - ISSUER_SERIAL
 - BST_DIRECT_REFERENCE
 - X509_KEY_IDENTIFIER
 - SKI_KEY_IDENTIFIER
 - KEY_VALUE
 - USER_NAME_TOKEN
 - CUSTOM_SYMM_SIGNATURE

In addition, the following identifiers are available when the signature is

based on a UsernameToken profile:

- USER_NAME_TOKEN
- CUSTOM_SYMM_SIGNATURE

- **User XPath part selection**

- Specify an XPath query that describes parts of the XML document that can be subjects of the algorithm. By default, the body is the subject. Click the **XPath Helper** button to build the Xpath expression.

- **Key**

- Select the key used for the encryption. The details of each key vary.
 - **x509 key**: This specifies the name and password of the x509 key and the keystore where it is located.
 - **User name token key**: This specifies a user name and password for the signature.
 - **Encrypted key**: This specifies a reference to an encrypted key that was previously defined in the security stack. Click **Insert a new encrypted key** to create a new encrypted key definition block.

- **Signature algorithm name**

- Specify the signature method algorithm as described in the XML Signature Syntax and Processing specification.

- **Canonicalization**

- Specify the canonicalization method to be used as described in the XML Signature Syntax and Processing specification.

- **Inclusive namespaces**

- Specify whether the canonicalization is exclusive as described in the Exclusive XML Canonicalization specification.

- **Custom Security Algorithm**

- If you want to use a Java™ class as a custom security algorithm, then use this stack element to apply the custom algorithm to the service.

- **Java Project**

- If you have not implemented a custom Java class, select **Java Project**, type a name for the new project, and click **Generate** to create a new Java class with the default structure for custom security implementations. **Note:** If you are using IBM® Security AppScan®, this field is not available.

- **Implementation class**

- Specify the name of the class that implements the custom security algorithm. Click **Browse Class** to select an existing Java class from the workspace.

- **Properties**

- Use this table to send any specific properties and associated values to the custom security algorithm.

Related information:

[Generic service client reference](#)

Testing web services with sample JSPs

After you create a web service, you can generate sample JSPs that can be run on the server to test the web service.

About this task

To generate a sample JSP to test a web service:

Procedure

1. Select your Java™ proxy bean in the Enterprise Explorer, right-click, and select **Web services**.
 - If you are testing a JAX-WS web service select **Generate JAX-WS JSPs**.
2. On the Web Service Client Test page, you have the following options:
 - Test the generated proxy: If selected, the sample client is launched in a web browser so you can see whether the proxy works.
 - Select your test facility. Currently the only available option is to generate web service sample JSPs.
 - Folder: Select the preexisting folder where the JSP is located.
 - Server: Select the server that you want to use.
 - Server instance: Select an existing instance of this server. If one does not exist, the wizard creates it for you.
 - Methods: Select the methods to expose in the JSP.
 - Run test on server: Select this option to start the server automatically.
3. The proxy is launched in a web browser at the following URL: `http://localhost:port/WebProjectClient/sampleBeanName/TestClient.jsp`. If you changed the folder location in the previous step the value of `sampleBeanName` reflects the new location.
4. You can use this sample application to test the web service by selecting a method, entering a value for the method, and clicking **Invoke**. The result of the method displays in the Results pane.
 - Setting endpoints: If you do not know the port that WebSphere® Application Server uses, it can be found in the administrative console at **Servers > Application servers > server_name > Configuration tab > Ports > WC_defaulthost**.
 - JAX-WS JSPs: To change the endpoint, edit the endpoint that is listed in the Endpoint field of the Quality of Service pane of the JSP, and click **Update**.
 - Asynchrony: If you are using JAX-WS JSPs and you selected to generate asynchronous methods for your proxy, you can enable asynchronous invocation in the `TestClient.jsp` file using the polling style. If you select to test the service asynchronously, when you invoke the business method, a new link displays, indicating that the method is in progress. Click the link to display the method response in the Results pane.
 - Bypassing JAXB mappings: Unlike JAX-RPC 1.1, JAX-WS does not support the generation of Service Endpoint Interfaces with business methods using SOAPElements. Selecting **Bypass JAXB and use XML payloads** replaces the JAXB bindings view of the `TestClient.jsp` methods with the raw SOAP message. If you want use this function, enter the required SOAP body entries for into the provided template.

Parent topic: [Testing and validating web services](#)

Related concepts:

[Testing and validating web services](#)

Related tasks:

[Testing WSDL documents and Web services with the WSDL Explorer](#)

Limitations of web services

This file contains a comprehensive list of limitations, both permanent and temporary, that affect web services.

The limitations that you might encounter when you work with web services are divided into the following sections:

- [Problems encountered when you use a secured WebSphere Application Server](#)
- [Problems encountered when you use the web services wizards](#)
- [Problems encountered when you use WebSphere JAX-WS runtime environment and JAXB](#)
- [Limitations when you create a web service client](#)
- [Limitations of the web service sample JSPs](#)
- [Web Services Explorer problems](#)
- [Problems when you use web services command-line tool](#)

Problems encountered when you use a secured WebSphere Application Server

Important: Applicable edition: Full profile

- You might encounter the following error message: `SSLSocketFactory error message: java.net.SocketException:`

```
java.lang.ClassNotFoundException: Cannot find the specified class
com.ibm.websphere.ssl.protocol.SSLSocketFactory
```

Before you use the Web Services Explorer, generating sample JSPs, creating a web service or client from an HTTPS WSDL file, or deploying an Axis web service, you must configure the workbench to communicate with a server via Secure Sockets Layer (SSL). Information about this task can be found here: [Problems working with a secured server using SSL connections](#).

- You might encounter the following error message: `Error opening socket: javax.net.ssl.SSLHandshakeException:`
`unknown certificate`

To use the web services wizard to retrieve an HTTPS WSDL or to use the Web Services Explorer against a secured WebSphere® Application Server, complete the steps in [Configuring the IBM® JRE to talk to a secured WebSphere Application Server](#).

Problems encountered when you use the web services wizards

- If you click **Cancel** part way through any of the web services wizards, you might have web service entries in the deployment descriptors and generated files in your workspace. Manually delete these files after you close the wizard.
- Web service wizards fail to publish EARs: Problem: When a remote server is created by using the stub run time, the EAR is not published.
Cause: The web service wizard has no way to find out whether a server can be started or not.
Solution: In any perspective where the Servers view is available, right-click the remote server and select **Add and Remove Projects**. Select the project and add it to the server.
- Web service wizards unable to consume the chosen project type: Problem: If you attempt to run the web service wizards on an EAR that contains a Java EE application client project with the **Install service on server** option selected, or an EJB project that does not contain an EJB, you cannot use the wizard to generate the web service.
Cause: The web service wizards check to ensure that they can consume the type of project that is selected on page three of the wizard. If they cannot, the wizard prevents you from continuing.
Solution: If you cancel of the wizard and rerun with **Install service on server** not selected, or add an EJB to the EJB project, you can progress in the wizard.
- If a WSIL file is selected in the workspace and then the web services wizard is launched, the wizard states that the selection is invalid because a WSDL file must be selected from the WSIL file. To resolve this, click **Browse** in the web services wizard to select a specific WSDL file.

- The web services wizard might produce an error during creation of a web service if the name of the target web project includes non-latin-1 characters. The problem occurs because the web services code generators calculate HTTP URLs based on the project name, and non-Latin-1 characters are not permitted in HTTP URLs. To resolve this, use project names that contain only latin-1 characters.
- If you have more than one EAR project in the workspace, and you select Enterprise JavaBeans in an EJB project and launch the web service wizard the selected enterprise bean might not be selected when the wizard launches. In the Select Service Implementation dialog box, the EAR project that is selected is the first EAR in the workspace alphabetically, not the EAR associated with the EJB project that contains the bean. To resolve this in the Select Service Implementation dialog box, manually select the wanted EAR project.
- When you select to monitor web services traffic on the first page of the web services wizard, occasionally the TCP/IP Monitor view does not display automatically. You can open the view manually by selecting **Windows > Show view > Debug > TCP/IP Monitor**. The request and response for the web service is shown on that view.

Problems encountered when you use WebSphere JAX-WS runtime environment and JAXB

- If you try to create a schema library where there are different schema documents that use the same namespace, the function generates projects and code with compilation errors. The compilation errors are in the `ObjectFactory.java` file. The problem is that, for a schema, the JAXB code is generated into a package whose name is based on the `targetNamespace` of the schema. An `ObjectFactory.java` class is also generated into this package and its contents are based on all of the schemas of the same namespace. It cannot be split or duplicated to resolve the compilation errors because the runtime environment expects this one class per package. Thus the schema library feature works only if all the schema dependencies are in different `targetNamespaces`.
- New in JAXB, the IBMWebSphere JAX-WS runtime environment is no longer responsible for resolving name collisions. When these collisions occur, you might see an error message such as `error: Property "property_name" is already defined`. The following tutorial provides more information and some examples of how to work around naming collisions: [The Java™ web services Tutorial: Customizing JAXB Bindings](#)

Limitations when you create a web service client

- When you deploy a web service client to an EJB project, on the proxy configuration page you must pick the EJB to which the web service client is scoped. This selection is a requirement defined by JSR-109 where the `component-name` of a `component-scoped-refs` element must match the `ejb-name` of an EJB in the module. When there is at least one EJB in the project, no errors occur. However, if there are no EJBs in the project, the options on the proxy configuration page are editable and have a default value equal to the service name from the WSDL suffixed by "EJB". Furthermore, the client wizard displays a warning that indicates that the deployment is incomplete because the web services client is not scoped to an existing EJB. You can fix this error by creating an EJB with the same name as the `component-name` used in `webservicessclient.xml`. However, if you choose to end the client creation by canceling from the wizard, the wizard does not remove the incomplete `component-scoped-refs` from `webservicessclient.xml`. Remove the incomplete `component-scoped-refs` manually before redeployment.

Limitations of the web service sample JSPs

- Web service sample JSPs have no support for methods that take in arrays or collections as parameters. Such methods are omitted from the sample JSP's Methods pane.

Web Services Explorer problems

- When the Web Services Explorer loads WSDL files that use multiple inline schemas, warning messages are generated for types that are referenced across these schemas. The warning message is similar to: `Reference of type <qualified_type_name> isn't resolved`. These are warnings and not errors and do not interfere with the Web Services

Explorer's ability to start operations, and therefore can be safely ignored.

- The Web Services Explorer does not support invocation of web services by authenticating proxy servers that use a protocol other than HTTP Basic Authentication, such as NTLM. If the Web Services Explorer fails to connect to the chosen web service for this or any other reason, it displays in its results pane the error `IWAB0135E An unexpected error has occurred` followed by information about the failure as retrieved from the HTTP response. To resolve this issue, switch to an HTTP Basic Authenticating proxy server.
- Web Services Explorer has problems when you switch from the source view to the form view if the schema uses choices that contain sequences. If you are using a choice element in a `complexType`, the choice contains two or more elements including a sequence, and you switch from the form view to the source view, the Web Services Explorer might be unable to correctly convert the corresponding XML fragment in the source view back into the form view without losing data. To resolve this problem, start the sequence from the form view when the data is initially entered. When this configuration occurs, it is not recommended the source view be used. If the source view is used, do not switch back to the form view.
- The Web Services Explorer imports WSDL without resolving relative import URLs. When you use the Web Services Explorer, there is an issue when you import a WSDL from the WSDL Page to the workspace. If the option **Import as a service reference to a WSIL document** is chosen this refers to the original URL of the WSDL and therefore all relative imports are reachable. If **Import WSDL Document** is chosen and the WSDL file contains relative imports, these imports are no longer reachable, leaving the copy of the WSDL difficult or impossible to process elsewhere in the tools. If there are relative imports in the WSDL you are importing, use the WSIL method. If there is a need for a copy of the WSDL, manually check for relative imports after the import to resolve any broken links.
- **8.5.5.5** The Web Services Explorer does not work if you use the Java 8 version of the Java virtual machine (JVM) in Eclipse. If you start Web Services Explorer with Java 8, the main window displays `HTTP Error 500`. To resolve this issue, use Java 7 as the Eclipse JVM, or use the generic service client.

Problems when you use web services command-line tool

- Importing an EAR containing an EJB Client generated by the command-line tools using Java EE 1.4 into the workspace results in compilation errors. To fix the errors, right-click the EJB project, and select **Properties**. Go to **Java Build Path**, and select the **Libraries** tab. Remove the `EJBClientProject/imported_classes(class folder)` entry. Add class folder `EJBServiceClient/imported_classes/Meta-inf/classes`. Click **OK**.
- [Full profile] After you import an EAR containing an Application Client generated by the command-line tools using Java EE 1.4 into the workspace, you get a `ClassNotFoundException` error when you run the client. To fix the errors, right-click the Application Client project, select **Properties**. Go to **Java Build Path**, and select the **Libraries** tab. Remove the `ApplicationClientProject/imported_classes(class folder)` entry. Add class folder `ApplicationClientProject/imported_classes/Meta-Inf/classes`. Click **OK**.
- The Web Services Ant tasks that target the Axis runtime handle the generation of web services but not the deployment of them. As a result, Axis web services that are created using the Ant tasks are not reachable on the running server until you manually deploy your services with the Apache Axis native **AdminClient** utility. Deploy the generated `deploy.wsdd` file in subdirectories under the `WebContent/WEB-INF` directory of the web project to Axis manually using the Axis AdminClient tool. Refer to the *Using the AdminClient* section of the Axis 1.4 User's Guide for more detail.

Parent topic: [Developing web service applications](#)

Developing Java batch applications

You can develop batch applications that are based on Java™ Specification Request (JSR) 352, and then submit Java batch jobs to run on a Liberty server.

- [Java batch overview for WebSphere Developer Tools](#)

With the Java batch feature, you can run applications that are computationally intensive or that have high volumes of records to process. You can submit jobs for these applications from WebSphere® Application Server Developer Tools for Eclipse so that the applications run on a Liberty server.

- [Installing Java EE batch tools](#)

You can install the optional Java EE batch tools onto WebSphere Developer tools on a computer with or without Internet access. After you install this component and WebSphere Developer tools, you can then develop Java batch applications and submit them to run on WebSphere Application Server Liberty Profile.

- [Job Specification Language editor](#)

You can use the Job Specification Language (JSL) editor to modify the contents of the xml file for your Java batch job.

- [Creating a Java batch project with the Java EE Batch Project wizard](#)

You can create a Java batch project with the Java EE Batch Project wizard so that you can create and submit Java batch jobs.

- [Creating a Java batch job with the Batch Job wizard](#)

You can use the Batch Job wizard in your WebSphere Application Server Developer Tools for Eclipse environment to create a Java batch job. You can run computationally intensive applications or transactional applications with your job.

- [Submitting a Java batch job](#)

You can submit a job for your Java batch application and run that Java batch application on a Liberty server.

- [Viewing Java batch job logs for WebSphere Developer Tools](#)

When you run Java batch jobs in WebSphere Developer Tools, a log is created for each job. You can view these logs after you submit a job, or you can retrieve them later.

8.5.5.6 Java batch overview for WebSphere Developer Tools

With the Java™ batch feature, you can run applications that are computationally intensive or that have high volumes of records to process. You can submit jobs for these applications from WebSphere® Application Server Developer Tools for Eclipse so that the applications run on a Liberty server.

The Java batch feature in WebSphere Application Server Developer Tools for Eclipse is optional. This feature provides tools to support development of applications that are based on [Java Specification Request \(JSR\) 352](#). The Java batch feature is also part of the Liberty runtime environment. For information on this feature in the Liberty runtime environment, see the topic on [configuring the liberty profile for Java batch](#).

The basic steps that you go through to submit a job for your Java batch application and have the application run on a Liberty server are as follows:

1. [Install the optional Java EE batch tools for WebSphere Application Server Developer Tools for Eclipse](#).
2. [Create a Java batch project](#).
3. [Create a Java batch job in your Java batch project](#).
4. [Package your Java batch application](#).
5. [Deploy the web archive \(WAR\) file that contains the Java batch application to a Liberty server](#).
6. [Submit your Java batch job so that it runs on a Liberty server](#).

Parent topic: [Developing Java batch applications](#)

Installing Java EE batch tools

You can install the optional Java™ EE batch tools onto WebSphere® Developer tools on a computer with or without Internet access. After you install this component and WebSphere Developer tools, you can then develop Java batch applications and submit them to run on WebSphere Application Server Liberty Profile.

Before you begin

Important: Applicable edition: Liberty profile

You need to install WebSphere Application Server Developer Tools for Eclipse and WebSphere Application Server Liberty Profile Developer Tools. If only the former feature is installed, you can create batch jobs, but you cannot submit them or target them to a server.

To install Java EE batch tools on a computer without Internet access, you need another computer with Internet access to download a .zip file. That file can then be moved to the computer without Internet access to complete the installation.

About this task

Add the optional Java EE batch tools to your installation of WebSphere Application Server Developer Tools for Eclipse.

Procedure

- Installing Java EE batch tools on a computer with Internet access

1. Select **Help > Install WebSphere Software**. The WebSphere Software Installer wizard displays.
2. Click **Install** for the **IBM Java EE Batch** component, and then click **Finish**.
3. On the **Install Details** page, review the items that you want to install and click **Next**.
4. On the **Review Licenses** page, review the license text.
5. If you agree to the license terms, click **I accept the terms of the license agreement**, and then click **Finish**. The installation process starts.
6. When the installation process completes, restart the workbench.

- Installing Java EE batch tools on a computer without Internet access

1. Download the .zip file for WebSphere Application Server Developer Tools for Eclipse to a directory on your computer, and then move that file to the computer without Internet access. The .zip file can be found at <https://developer.ibm.com/wasdev/downloads/>.
2. If you agree to the license terms, click **I accept the terms of the license agreement**, and then click **Finish**.
3. Open Eclipse and click **Help > Install new software**.
4. Click **Add > Archive** and select the .zip file that you downloaded.
5. Optional: Enter a name for the repository.
6. Clear **Group items by category**.
7. In the **Filter text** field, enter `batch`.
8. Select **IBM Java EE Batch** and click **Finish**.

Parent topic: [Developing Java batch applications](#)

Related tasks:

[Installing WebSphere Application Server Developer Tools for Eclipse](#)

Job Specification Language editor

You can use the Job Specification Language (JSL) editor to modify the contents of the xml file for your Java™ batch job. The JSL editor has two views for your batch job, the Design view and the Source view. You can click the corresponding tabs to move between the Design view and the Source view.

Design view

The Design view has two sections, **Overview** and **Details**.

- Overview

- Specifies a tree view of your job. You can add and remove items in the tree such as steps, listeners, and properties. You can also reorder the items in the tree.

- Details

- Specifies attributes of an item in the tree. You can modify these attributes. The attributes vary from item to item. Left-click an item in the tree to display the attributes of the item.

Source view

In the source view you can view and edit the xml source for your job. You can use content assist when you edit the source for your job.

- Content assist

- Helps you finish a line of code in an xml file that you display in the JSL editor. The placement of the cursor in the source file provides the context for the content assist to offer suggestions for completion.

- Job XML substitution in the JSL editor

You can use job XML substitution in the Job Specification Language (JSL) editor to specify an expression for almost any attribute in your Java batch job. At run time, the expression resolves to a particular value.

Parent topic: [Developing Java batch applications](#)

Job XML substitution in the JSL editor

You can use job XML substitution in the Job Specification Language (JSL) editor to specify an expression for almost any attribute in your Java™ batch job. At run time, the expression resolves to a particular value.

Your Java batch job is defined in an XML file. You can display this file and work with it in the JSL editor. You can add various elements to your job, including step, decision, flow, listener, property, and split elements. To these elements you can add various attributes. The attributes that you can add vary from element to element. For various attributes, you can use job XML substitution.

Elements for job XML substitution

The job XML substitution expression consists of the following elements.

- Job operator

- Specifies where the value for the expression of an attribute comes from at run time. The valid values are `Job Parameters`, `System Properties`, `Job Properties`, or `Partition Plan`.

- Target

- Specifies the name of the property that is used to resolve the expression.

- Default value

- Specifies a default value to use if at run time the expression is not resolved, for instance, if the property is not found.

Attributes that have job XML substitution for specific elements

Attributes are in the JSL editor on the **Details** tab of an element. For those attributes that have job XML substitution, the **Job XML Substitution** button is next to the attribute on the **Details** tab. The following table lists elements and their attributes where you can launch the job XML substitution dialog. *Table 1. Elements with their attributes that can have job XML substitution*

Elements	Attributes
Property	Value
Flow	Next
Next	On*To*
End	On*
Fail	On*
Stop	On*Restart
Step	NextStart limit
Chunk	Item countRetry limitSkip limitTime limit

The Job XML Substitution dialog

When you click the **Job XML Substitution** button, the **Job XML Substitution** dialog is displayed. The dialog consists of the following fields:

- Operator

- Target

- Default

- Value

The **Job XML Substitution** dialog is pre-populated with valid values. What the values are depends on the job operator that you select for the **Operator** field.

- The values for the `Job Parameters` job operator are obtained from existing runtime configurations. **Important:** To provide a list of valid values for the `Job Parameters` value, you must have a launch configuration for the associated job.
- The values for the `System Properties` job operator are obtained from existing runtime configurations. **Important:** To provide a list of valid values for the `System Properties` job operator, you must have the following setup:
 - You must have a launch configuration for the associated job.
 - You must have a target Liberty server that is started.
 - If the target Liberty server is a local server, the target Liberty server must have a local connector.

- The `Partition Plan` job operator has no values.

The **Value** field is pre-populated with the current value of a particular attribute. You can edit the value or you can insert a substitution expression by clicking **Insert into value**.

Parent topic: [Job Specification Language editor](#)

8.5.5.6 Creating a Java batch project with the Java EE Batch Project wizard

You can create a Java™ batch project with the Java EE Batch Project wizard so that you can create and submit Java batch jobs.

Before you begin

Install the Java batch feature for WebSphere® Application Server Developer Tools for Eclipse. This feature is an optional. You can install it from **Help > Install WebSphere Software**.

Procedure

1. Select **File > New > Other > Java EE Batch > Java EE Batch Project**. The Java EE Batch Project wizard displays.
2. In the **Project name** field, type the name of the Java batch project.
3. In the **Project location** section, either use the default location, which is selected by default, or specify the workspace location to store the project files.
4. In the **Target runtime** field, select the target runtime environment for the project. Use the following supported server for running your batch projects:
 - WebSphere Application Server Liberty profile
5. For the **Configuration** section, use the default configuration or select **Modify** to add more function to your project by adding more facets.
6. In the Dynamic Web project membership section, add the project to a web archive (WAR) file.
 - A. Ensure that **Add project to Dynamic Web project** is selected. The **Add project to Dynamic Web project** field is selected by default. You need to add your project to a Dynamic Web project to deploy the application later.
 - B. In the **Dynamic Web project name** field, choose the web project that you want to contain your batch project. You can select an existing web project, or create a new project by accepting the default name or entering a new one.
7. Optional: For the **Working sets** field, add the Java batch project to a working set.
 - A. Select **Add project to working sets**.
 - B. If there are no working sets in the **Working sets:** list, or you want a different working set, click **Select** and add working sets to the **Working sets:** list.
 - C. Select the working set from the **Working sets:** field.
8. Click **Next**.
9. On the Java page, accept the default source folder and output folder, or specify different folders, then click **Next**. The Java Batch page displays.
10. Accept the default for creation of a batch job folder. All your Java batch jobs for this project are stored in this folder.
11. Optional: Select the setting to generate the `batch.xml` file. The `batch.xml` file is used during batch artifact resolution. This file contains mappings of references names to batch artifact implementation class names.
12. Click **Finish**.

Results

You created a batch project for JSR 352.

What to do next

[Create a batch job.](#)

Parent topic: [Developing Java batch applications](#)

8.5.5.6 Creating a Java batch job with the Batch Job wizard

You can use the Batch Job wizard in your WebSphere® Application Server Developer Tools for Eclipse environment to create a Java™ batch job. You can run computationally intensive applications or transactional applications with your job.

Before you begin

Complete the following tasks:

- Install the Java batch feature for WebSphere Application Server Developer Tools for Eclipse. This feature is an optional. You can install it from **Help > Install WebSphere Software**.
- Create a Java batch project.

Procedure

1. In the Enterprise Explorer view, select your Java batch project. When the New Batch Job wizard displays in the next step, the **Batch Project** field is pre-populated with the name of your Java batch project.
2. In the Enterprise Explorer view, select **File > New > Other > Java EE Batch > Batch Job**. The New Batch Job wizard displays.
3. In the **Job Name** field, type the name of the batch job.
4. If you want to allow this job to be restarted when it is running, then leave **Restartable** selected; otherwise, clear **Restartable**.
5. Click **Finish**. An xml file with the same name as the batch job displays in the Job Specification Language (JSL) editor.
6. Populate your Java batch job with Java batch artifacts. Choose one or more of the following scenarios to add Java batch artifacts to your Java batch job.
 - [Adding a batchlet step to a Java batch job](#)
 - [Adding a chunk step to a Java batch job](#)

Results

You created a batch job that contains steps such as batchlet steps and chunk steps so that you can run computationally intensive applications or transactional applications.

- [Adding a batchlet step to a Java batch job](#)

You can add a batchlet step to the xml file of a Java batch job. Use a batchlet step to perform task-oriented processing such as running a command or transferring a file.

- [Adding a chunk step to a Java batch job](#)

You can add a chunk step to the xml file of a Java batch job. You can read, process, and write individual items in the chunk step, and also restart the chunk step.

Parent topic: [Developing Java batch applications](#)

Adding a batchlet step to a Java batch job

You can add a batchlet step to the xml file of a Java™ batch job. Use a batchlet step to perform task-oriented processing such as running a command or transferring a file.

Before you begin

A batchlet step is started once, runs to completion, and returns an exit status. Unlike a chunk step, the batchlet step does not read, process, and write individual items, which are data records.

Complete the following tasks:

- Install the Java batch feature for WebSphere® Application Server Developer Tools for Eclipse. This feature is optional. You can install it from **Help > Install WebSphere Software**.
- Create a Java batch project.
- Create a Java batch job in the Java batch project.

About this task

Use the Job Specification Language (JSL) editor to create the batchlet step. Then, create a batchlet class and add it to the batchlet step to implement the step.

Procedure

1. In the Job Specification Language (JSL) editor, open the xml file for your Java batch job. The JSL editor opens automatically if you click **Finish** when you create a Java batch job. An xml file with the same name as the batch job displays in the JSL editor.

If you are adding a step to an existing Java batch job xml file, complete the following steps to open the xml file in the JSL editor:

 - A. In the Enterprise Explorer view, select your Java batch project that you created.
 - B. Select **src > META-INF > batch-jobs**.
 - C. Double-click the xml file for your Java batch job. The file displays in the JSL editor.
2. Select the job node on the **Overview** tab; click **Add**.
3. Select **Step**; click **OK**.
4. Optional: Change the ID in the **ID*** field of the **Details** tab.
5. Select the step node on the **Overview** tab; click **Add**.
6. Select **Batchlet**; click **OK**.
7. Include a batchlet class to implement the batchlet step. You can create a new batchlet class for your project, use an existing batchlet class within your project, or use an existing batchlet class in another project that is in the class path of your project. Complete one of the following tasks:
 - A. Create a class to implement this batchlet step.
 1. Click the reference link on the **Details** tab. The New Batchlet Class wizard displays.
 2. In the **Java package** field, specify the name of the package.
 3. In the **Class name** field, specify the name of the class.
 4. Optional: In the **Super class** field, specify the name of the super class.
 5. Optional: In the **Interfaces** field, add more interfaces to be implemented by this class.
 6. For **Batch artifact loading options**, accept the default value of **Use the fully qualified name of the class**.
 - If you select the **Use the fully qualified name of the class** option, the runtime environment treats the batch artifact reference as a class name and loads the reference through the thread context class loader. The loading is

done to resolve a batch artifact reference when the archive loader returns null for this reference.

- If you select the **Use context and dependency injection (CDI)** option, the runtime environment uses the CDI implementation-specific loader to resolve the batch artifact reference by using the bean name that you specify.
- If you select the **Add the class to the batch.xml file** option, the runtime environment uses the archive loader to resolve the reference by looking for the specified identifier in the `batch.xml` file.

B. Use an existing batchlet class to implement the batchlet step.

1. Click **Browse** on the Details tab.
2. Search for classes by name by selecting **Search for classes** and then typing the class name.
3. Select the class name from **Matching items**.
4. As an alternative to the search, if a `batch.xml` file exists, select **Browse entries in batch.xml** and then select one of the entries from the table.
5. Click **OK**.

8. Click **Finish**.

Results

You created a batchlet step for a Java batch job. You included a batchlet class in the batchlet step so that you can implement the step in the Java batch job. You created the batchlet class or used an existing one.

Parent topic: [Creating a Java batch job with the Batch Job wizard](#)

Adding a chunk step to a Java batch job

You can add a chunk step to the xml file of a Java™ batch job. You can read, process, and write individual items in the chunk step, and also restart the chunk step.

Before you begin

Complete the following tasks:

- Install the Java batch feature for WebSphere® Application Server Developer Tools for Eclipse. This feature is an optional. You can install it from **Help > Install WebSphere Software**.
- Create a Java batch project.
- Create a Java batch job in the Java batch project.

About this task

Use the Job Specification Language (JSL) editor to create the chunk step. You will implement a reader node and a writer node for the chunk step. You can optionally create a processor node to process the input items. You can also optionally checkpoint the chunk step to restart it.

Procedure

1. In the Job Specification Language (JSL) editor, open the xml file for your Java batch job. The JSL editor opens automatically if you click **Finish** when you create a Java batch job. An xml file with the same name as the batch job displays in the JSL editor.
If you are adding a step to an existing Java batch job xml file, complete the following steps to open the xml file in the JSL editor:
 - A. In the Enterprise Explorer view, select your Java batch project that you created.
 - B. Select **src > META-INF > batch-jobs**.
 - C. Double-click the xml file for your Java batch job. The file displays in the JSL editor.
2. Select the job node on the **Overview** tab; click **Add**.
3. Select **Step**; click **OK**.
4. Optional: Change the ID in the **ID*** field of the **Details** tab.
5. Select the step node on the **Overview** tab; click **Add**.
6. Select **Chunk**; click **OK**. On the overview tab of the JSL editor, the chunk node displays as a child of the step. A reader node and a writer node display as children of the chunk node.
7. Implement the reader node.
 - A. On the Overview tab, click the reader node. The Details tab for the reader node displays.
 - B. Create or include a reader class to implement the reader node. You can create a reader class for your project, include a reader class from another step within your project, or include a reader class from a chunk step in another project. Complete one of the following tasks:
 - Create a reader class for your chunk step.
 1. Click the reference link on the **Details** tab. The Create Item Reader Class wizard displays.
 2. In the **Java package** field, specify the name of the package.
 3. In the **Class name** field, specify the name of the class.
 4. Optional: In the **Super class** field, specify the name of the super class or select **Use abstract Java Batch class as superclass**.

5. Optional: In the **Interfaces** field, add more interfaces for the class to implement.
6. For **Batch artifact loading options**, accept the default value of **Use the fully qualified name of the class**.
 - If you select the **Use the fully qualified name of the class** option, the runtime environment treats the batch artifact reference as a class name and loads the reference through the thread context class loader. The loading is done to resolve a batch artifact reference when the archive loader returns null for this reference.
 - If you select the **Use context and dependency injection (CDI)** option, the runtime environment uses the CDI implementation-specific loader to resolve the batch artifact reference by using the bean name that you specify.
 - If you select the **Add the class to the batch.xml file** option, the runtime environment uses the archive loader to resolve the reference by looking for the specified identifier in the `batch.xml` file.
7. Click **Finish**.
 - Use an existing reader class to implement the chunk step.
 1. Click **Browse** on the Details tab.
 2. Search for classes by name by selecting **Search for classes** and then typing the class name.
 3. Select the class name from **Matching items**.
 4. As an alternative to the search, if a `batch.xml` file exists, select **Browse entries in batch.xml** and then select one of the entries from the table.
 5. Click **OK**.
8. Implement the writer node.
 - A. On the Overview tab, click the writer node. The Details tab for the writer node displays.
 - B. Create or include a writer class to implement the writer node. You can create a writer class for your project, include a writer class from another step within your project, or include a writer class from a chunk step in another project.Complete one of the following tasks:
 - Create a writer class for your chunk step.
 1. Click the reference link on the **Details** tab. The Create Item Writer Class wizard displays.

The following descriptions about the Create Item Writer Class wizard are the same as those descriptions for the Create Item Reader Class wizard. The information is repeated here in context.
 2. In the **Java package** field, specify the name of the package.
 3. In the **Class name** field, specify the name of the class.
 4. Optional: In the **Super class** field, specify the name of the super class or select **Use abstract Java Batch class as superclass**.
 5. Optional: In the **Interfaces** field, add more interfaces for the class to implement.
 6. For **Batch artifact loading options**, accept the default value of **Use the fully qualified name of the class**.
 - If you select the **Use the fully qualified name of the class** option, the runtime environment treats the batch artifact reference as a class name and loads the reference through the thread context class loader. The loading is done to resolve a batch artifact reference when the archive loader returns null for this reference.
 - If you select the **Use context and dependency injection (CDI)** option, the runtime environment uses the CDI implementation-specific loader to resolve the batch artifact reference by using the bean name that you specify.
 - If you select the **Add the class to the batch.xml file** option, the runtime environment uses the archive loader to resolve the reference by looking for the specified identifier in the `batch.xml` file.
 7. Click **Finish**.
 - Use an existing writer class to implement the chunk step. The following steps are the same as those described in the reader class previously. The information is repeated here in context.
 1. Click **Browse** on the Details tab.
 2. Search for classes by name by selecting **Search for classes** and then typing the class name.
 3. Select the class name from **Matching items**.
 4. As an alternative to the search, if a `batch.xml` file exists, select **Browse entries in batch.xml** and then select one of the entries from the table.

5. Click **OK**.
9. Optional: Implement the processor node for the chunk node. The processor node is used to process inputs to the chunk step.
 - A. Select the chunk node on the **Overview** tab; click **Add**.
 - B. Select **Processor**; click **OK**. On the Overview tab of the JSL editor, the processor node displays as a child of the chunk node.
 - C. Click the reference link on the **Details** tab. The Create Item Processor Class wizard displays
The following descriptions about the Create Item Processor Class wizard are the same as those descriptions for the Create Item Reader Class wizard. The information is repeated here in context.
 - D. In the **Java package** field, specify the name of the package.
 - E. In the **Class name** field, specify the name of the class.
 - F. Optional: In the **Super class** field, specify the name of the super class.
 - G. Optional: In the **Interfaces** field, add more interfaces for the class to implement.
 - H. For **Batch artifact loading options**, accept the default value of **Use the fully qualified name of the class**.
 - If you select the **Use the fully qualified name of the class** option, the runtime environment treats the batch artifact reference as a class name and loads the reference through the thread context class loader. The loading is done to resolve a batch artifact reference when the archive loader returns null for this reference.
 - If you select the **Use context and dependency injection (CDI)** option, the runtime environment uses the CDI implementation-specific loader to resolve the batch artifact reference by using the bean name that you specify.
 - If you select the **Add the class to the batch.xml file** option, the runtime environment uses the archive loader to resolve the reference by looking for the specified identifier in the `batch.xml` file.
 - I. Click **Finish**.
 10. Optional: Checkpoint the chunk node so that you can restart the node. You can either checkpoint the chunk node after a specified number of records in the chunk are processed or according to a checkpoint algorithm.
 - A. Checkpoint the chunk node after a specified number of records in the chunk are processed.
 1. Select **Item** for the checkpoint policy
 2. Specify the number of items to process per chunk by providing a value for **Item count**.
 3. Specify the amount of time that passes before a checkpoint is taken by providing a value for **time-limit**.
 - B. Checkpoint the node that is based on a checkpoint algorithm.
 1. Select **Custom** for the checkpoint policy.
 2. On the Overview tab, click **Add**.
 3. Select **Checkpoint algorithm**; click **OK**. The Details tab for **Checkpoint algorithm** displays.
 4. Create or include a checkpoint algorithm class to implement the checkpoint algorithm. You can create a new checkpoint algorithm class for your chunk step, include a checkpoint algorithm class from another step within your project, or include a checkpoint algorithm class from a checkpoint algorithm step in another project. Complete one of the following sets of steps:
 - Create a checkpoint algorithm class to implement the checkpoint algorithm.
 - A. Click the reference link on the **Details** tab. The Create Checkpoint Algorithm Class wizard displays.
The following descriptions about the Create Checkpoint Algorithm Class wizard are the same as those descriptions for the Create Item Reader Class wizard. The information is repeated here in context.
 - B. In the **Java package** field, specify the name of the package.
 - C. In the **Class name** field, specify the name of the class.
 - D. Optional: In the **Super class** field, specify the name of the super class.
 - E. Optional: In the **Interfaces** field, add more interfaces for the class to implement.
 - F. For **Batch artifact loading options**, accept the default value of **Use the fully qualified name of the class**.
 - If you select the **Use the fully qualified name of the class** option, the runtime environment treats the batch artifact reference as a class name and loads the reference through the thread context class loader. The

loading is done to resolve a batch artifact reference when the archive loader returns null for this reference.

- If you select the **Use context and dependency injection (CDI)** option, the runtime environment uses the CDI implementation-specific loader to resolve the batch artifact reference by using the bean name that you specify.
- If you select the **Add the class to the batch.xml file** option, the runtime environment uses the archive loader to resolve the reference by looking for the specified identifier in the `batch.xml` file.

G. Click **Finish**.

- Use an existing checkpoint algorithm class to implement the checkpoint algorithm. The following steps are the same as those described in the reader class previously. The information is repeated here in context.

A. Click **Browse** on the Details tab.

B. Search for classes by name by selecting **Search for classes** and then typing the class name.

C. Select the class name from **Matching items**.

D. As an alternative to the search, if a `batch.xml` file exists, select **Browse entries in batch.xml** and then select one of the entries from the table.

E. Click **OK**.

11. Click **Finish**.

Results

You created a chunk step for a Java batch job. You implemented the reading and writing of items for the chunk step and optionally implemented processing of the input items. You also optionally implemented checkpoints for the chunk step so that you can restart the step.

Parent topic: [Creating a Java batch job with the Batch Job wizard](#)

Submitting a Java batch job

You can submit a job for your Java™ batch application and run that Java batch application on a Liberty server.

Before you begin

Complete the following steps:

1. Install the Java batch feature for WebSphere® Application Server Developer Tools for Eclipse. This feature is optional.
You can install it from **Help > Install WebSphere Software**.
2. Create a Java batch project.
3. Create a Java batch job in the Java batch project.
4. Ensure that your Java batch application is packaged. The wizard automatically packages the resulting batch project in a web archive (WAR) file, but you can also package your project in other resources.
5. [Create a Liberty server](#) or [create a remote Liberty server](#) and [configure it for Java batch applications](#).
6. Deploy the resource that contains the batch application to a Liberty server.
 - A. Select **Window > Show View > Servers**.
 - B. Right-click the Liberty server and select **Add and Remove...**. The Add and Remove wizard displays.
 - C. Select the resource that you want to deploy from the Available area to the Configured area by clicking **Add >**; then, click **Finish**.
7. Start the Liberty server.
 - A. Select **Window > Show View > Servers**.
 - B. Right-click the Liberty server and select **Start**.

Procedure

1. For Java batch projects, in the Enterprise Explorer view, select a Java batch project, and then select **src > META-INF > batch-jobs > batch_job.xml**.
2. Right-click the *batch_job.xml* file.
3. Select **Run As > Java EE Batch Job**.
4. Select the target server on which the Java batch job runs.
5. Specify the user ID and password of a user who is configured in your Liberty server. This user ID is used to submit Java batch jobs.
6. Optional: In the **Job parameters** section, enter any job parameters that you defined for your batch application.
 - A. Click **Add**.
 - B. The Add Parameter dialog displays.
 - C. Enter the name and value of the parameter.
 - D. Repeat the process to add any additional parameters.
7. Click **Apply** and then **Run**.

Results

You submitted a batch job to your Liberty server so that you can run your Java batch application.

If the job submission is successful, a dialog shows information about the job instance that you submitted, and job execution details.

If the submission fails, a dialog shows possible causes of the problem. If the information provided in the dialog is not sufficient to diagnose the problem, you can check the Error log view to see more details about the failure. You can also check the server logs, which are located in the `wlp\usr\servers\<server>\logs` directory.

Parent topic: [Developing Java batch applications](#)

8.5.5.6 Viewing Java batch job logs for WebSphere Developer Tools

When you run Java™ batch jobs in WebSphere® Developer Tools, a log is created for each job. You can view these logs after you submit a job, or you can retrieve them later.

About this task

View Java batch job logs immediately after you submit a job or at any time later.

Procedure

1. After you submit a job, select **Open the Batch job log view after closing this dialog** in the Submitted job dialog window. **Note:** This view shows job logs only for the most recently submitted job, and does not show logs from jobs that were previously submitted to the same server. To see all logs for all jobs in the locally started servers, click **Show all logs**.
2. Alternately, open the job log view at any time by clicking **Window > Show View > Java EE Batch > Java EE batch Job Logs**. This view displays the logs for all started servers that you added to a Java EE Batch job run configuration.
3. Double-click the log you want to view. The log opens in the internal browser or in the browser you previously set as the default. When you open the browser for the first time, you are prompted to enter your credentials.
4. Optional: Click **Refresh view** if you need to update the contents of the view without changing the current search filters.

Parent topic: [Developing Java batch applications](#)

Related information:

[Viewing Java batch job logs](#)

Testing and publishing on a server

You can create, configure, and manage servers so that you can test your project resources. After you complete your testing, you can publish your applications onto your servers.

Learn about testing and publishing on a server

You can test an application that is running on the development workbench before publishing the application to the directories of an application server. Publishing involves copying files (application, resource files, and deployment descriptor files) to the correct location for the server to find and use them. Read this page for information on how to integrate the workbench with a server.

Overview

You can read the following topics before testing and publishing an application on a server. They provide planning overview information that may be useful if you are new to working with a server in this development environment.

- **Resources used to test and publish on your server**

- Learn what is a server, server project, server runtime environment and server configuration. Learn what server resources you need to define in the workbench to communicate to a server. In addition, learn what types of servers are supported with this development workbench.

- **Servers view**

- Learn how you can manage the server within the workbench.

Getting started

If you are already familiar with administering your application server the following topics will help you set up your workspace for integrating your application server with the workbench, and guide you through the configuration and management process.

- **Using the Liberty profile as an application development environment**

- The Liberty profile provides an application development environment for your web and OSGi applications.

- **Defining the server runtime environments preferences**

- This task directs the workbench to use a specific runtime environment of an application server for compiling, testing, or running your application.

- **Creating a server**

- This task adds a server to the Servers view.

- **Configuring servers**

- If you are working with a WebSphere Application Server, this task sets up options and settings between the workbench and a WebSphere Application Server.

- **Managing servers**

- This task enables you to control a server in the Servers view, such as add a project to a server, stop and start a server and many more.

- **Testing artifacts on a server**

- This task describes the Run on Server option available in the workbench used to test application artifacts on a server.

Web resources for learning

See the following links for more information:

[WebSphere Application Server Information Center](#)

Note: [Latest developerWorks® articles and tutorials about *application servers*](#)

Overview: Testing and publishing on your server

The testing and publishing tools provides runtime environments where you can test JSP files, servlets, HTML files, web services, enterprise beans, Java™™ Persistence API, Java classes and many more artifacts.

You can use the workbench to test and publish resources from many types of projects, here are examples:

- Dynamic Web projects, which typically contain JSP files, HTML files, servlets, and JavaBeans
- Static Web projects, which typically contain HTML files and graphic files
- Enterprise Applications projects, which may contain Java Archive (JAR) files or Web Archive (WAR) files or both, and pointers to other Web or EJB projects
- EJB projects, which contain enterprise beans
- Application client projects

After testing your application, you can use the tools to publish the application.

Server Definitions

The workbench defines servers to test and publish your projects. Servers are definitions that identify where you want to test your projects. You can either have the development environment create the servers automatically for you, or you can create them using the New Server wizard (right-click in the Servers view and select **New > Server**).

Compatible Application Servers

The Version 8.5 release of the product introduces support for WebSphere® Application Server Liberty Profile. The Liberty Profile server is a lightweight profile of the application server for web, Web 2.0, mobile, and OSGi applications. The Liberty profile is a highly composable, fast to start, ultra lightweight profile of the application server that is optimized for developer productivity and smaller, simpler production server deployments. Use this server when you are building applications that do not require the full Java EE environment of traditional enterprise application server profiles. For more information about the Liberty profile server, see the [Liberty profile server](#) help topic.

The application server that is commonly used with the product is WebSphere Application Server, which is tightly integrated with the workbench. To test, run, and debug applications on the server, the workbench contains tools such as **Run On Server**. The following features are available from within the workbench:

- Starting tools from WebSphere Application Server, such as the administrative console and the Profile Management Tool
- Developing, running, and debugging administrative scripts against the application server

Server adapters for particular servers are included in the product from the Web Tools Platform that is based on Eclipse technology. You can also download more server adapters to the workbench: Open the Servers view, right-click; then click **New**. In the New Server wizard, click **Download additional server adapters**.

Integrating with WebSphere Application Server

The following versions of IBM® WebSphere Application Server are compatible with this product:

- V7.0 with the option to install the Feature Packs for Communications Enabled Applications, Modern Batch, OSGi Applications, Service Component Architecture, Web 2.0, XML, or all
- V8.0
- V8.5
- V8.5 Liberty Profile

With WebSphere Application Server, you can create the following types of servers:

- Stand-alone (unmanaged) server

- A node that is defined in the cell of a WebSphere Application Server topology that does not run a node agent to manage its process.

- Federated (managed) server

- A server that runs a node agent to manage its process.

The product does not support the tools for a federated (managed) WebSphere Application Server Network Deployment environment. The product workbench does not support connections to the deployment manager, and it does not support publishing to clusters or federated nodes. However, there is support for stand-alone (unmanaged) WebSphere Application Server Network Deployment; you can test, run, and debug applications from the workbench to a stand-alone (unmanaged) application server. For both managed and unmanaged WebSphere Application Server Network Deployment, you can continue to run the Profile Management Tool from within the workbench to create and augment profiles.

WebSphere Application Server Developer Tools for Eclipse does not provide support for embedded hypertext transfer protocol (HTTP) servers or web servers with IBM WebSphere Test Environment. These web servers can be added by using the WebSphere Application Server admin console. After being added, the web servers don't show after restarting WebSphere Application Server or re-publishing to WebSphere Application Server. This is because the embedded web servers are not supported by the server tools that provide tools to run WebSphere Application Server. For more information, see the DCF document at <http://www-01.ibm.com/support/docview.wss?uid=swg21064208>.

Specification-level

The workbench provides server selection options that are based on the specification-level that you defined for your project. The following list summarizes specification-level support for the different version-levels of the WebSphere Application Server. For more information about supported WebSphere Application Server specification-levels, see *Specifications and API documentation*. WebSphere Application Server Version 7.0

Java EE 5 (or earlier) support is available for running applications against a WebSphere Application Server V7.0.

WebSphere Application Server Version 8.0

Java EE 6 (or earlier) support is available for running applications against a WebSphere Application Server V8.0.

WebSphere Application Server Version 8.5

Java EE 6 (or earlier) support is available for running applications against a WebSphere Application Server Version 8.5 .

WebSphere Application Server Version 8.5 Liberty Profile

Java EE 6 (or earlier) support is available for running applications against a WebSphere Application Server Version 8.5 Liberty Profile.

Related information:

[Which test server will start](#)

[Keyboard shortcuts for Server Tools](#)

[Resources used to test and publish on your server](#)

When the test server requires restarting

The following subtopics describe different situations where you might need to restart the server.

In the development environment, you might want to change an application while it is running on a server, for example, if you are debugging an application on a server. In some cases, you can dynamically reload modified code without restarting the server. You might lose the state of the program, depending on the type of resource that is modified and the type of server.

When an application is running on a server and you change the code, the Java™ virtual machine keeps running the initial code until the code is reloaded automatically or manually. For example, you can modify JSP source and the changes will reload automatically on the server. For other resources, such as Java classes that are running on Tomcat, you must restart the server to ensure that the changes are recognized by the server.

Changes to server configuration

If you change the server or the server configuration while the server is running, for example, if you change the port number, you need to restart the server.

Changes to JSP, HTML, graphic, and files that are not Java

Any changes that you make to your JSP, HTML, GIF, JPG, or a similar resource file; when you save the file while the server is running, you must publish the application on the server. Issuing a publish is especially important if changes to the resource file are a linked resource, which might be files or folders that are stored in locations in the file system outside of the workspace. Changes to linked resources cannot be detected by the workbench or the server as the changes happened outside of the workspace.

To publish the application on the server, if the **Automatically publish when starting servers** check box (available from **Window > Preferences > Server > Launching** page) is enabled, you can wait for the automatic publishing interval to pass. However, if the **Automatically publish when starting servers** check box is clear, you must manually publish your application (right-click the server in the Servers view and select **Publish**). After a publish is complete, refresh the web browser for the server to recognize the changes. The state of the application will not be lost. **Tip:** At anytime during your development, if you notice that your change to a static resource is not updated on the server and the **Automatically publish when starting servers** check box is clear, try running a manual publish by right-click the server in the Servers view and select **Publish**. After the publish is complete, refresh the web browser and the changes will update on the server.

For more information on automatic or manual publishing to the server, see the [Publishing your application](#) topic.

Changes to servlets and related classes

If you change a servlet and save the file while the server is running, the servlet is reloaded, if you enable reloading for that application. When the server runs hot method replace¹, the changes take place automatically without needing to refresh the browser. The server recognizes the change when publishing the application on the server. If you refresh the web browser, the state of the application is not lost.

To publish the application on the server, you can wait for the automatic publishing interval to pass or manually publish your application (right-click the server in the Servers view and select **Publish**). For more information on automatically or manual publishing to the server, see the [Publishing your application](#) topic. The session data for that project is lost, but the state of other projects within the application is unchanged. You can restart the project from the menu in the Navigator view. If you are running Tomcat and have not enabled the reloading feature, you need to restart the server.

For WebSphere Application Server, the rules also apply to any dependent classes or deployment descriptors of the web project. If you modify the security or login configuration properties of the `web.xml` deployment descriptor that is running on WebSphere Application Server, you need to restart the server. For Tomcat, a restart of the server is required for any of these changes.

Adding servlets, classes, or JSP files

If you add a servlet, dependent class, or JSP file to a web project while the server is running, the changes are recognized if you enabled reloading. If you did not enable reloading, you need to publish the application and restart the EAR project if you are running WebSphere Application Server, or publish the application and restart the server if you are running Tomcat. To publish the application on the server, you can wait for the automatic publishing interval to pass or manually publish your application (right-click the server in the Servers view and select **Publish**). For more information on automatically or manually publishing to the server, see the [Publishing your application](#) topic. When the server runs the hot method replace function in debug mode, changes to Java classes are automatically recognized.

Changes to EJB resources

For WebSphere Application Server, the server dynamically restarts the EJB project in the EAR. When the server runs the hot method replace function in debug mode, changes to Java classes are automatically recognized.

Remember: Tomcat does not support EJB testing and publishing.

Changes to resources within an Enterprise Application project

For WebSphere Application Server, if you change any resource within an enterprise application project while it is running on the server, the server dynamically restarts the EAR. Tomcat does not support Enterprise Application project testing and publishing.

Table 1. Summary of actions to take when resources are modified while the server is running. Table 1 describes the actions a user needs to take when resources are modified while the server is running. [Liberty Core, full profile]

Resource modified		Required Action		
		WebSphere Application Server	Apache Tomcat	WebSphere Application Server Express®
Server configuration		Same behavior as the stand-alone WebSphere Application Server, for details refer to the Information Center for WebSphere Application Server	Restart the server	Restart the server
JSP		Refresh the web browser	Refresh the web browser	Refresh the web browser
Servlet	Reloading enabled	Refresh the web browser	Refresh the web browser	Refresh the web browser
	Reloading disabled	Restart the EAR project or the server	Restart the server	Restart the EAR project or the server
Dependent classes or deployment descriptors	Reloading enabled	Refresh the web browser	Refresh the web browser	Refresh the web browser

	Reloading disabled	Restart the EAR project or the server	Restart the server	Restart the EAR project or the server
EJB implementation		Automatically reloaded	Not supported	Not supported
EJB interface or interface's dependent classes		Automatically reloaded. Restart the application client if the application client holds a reference to the EJB.	Not supported	Not supported
EJB, dependent classes, or deployment descriptors		Automatically reloaded	Not supported	Not supported
EJB or dependent class added		Automatically reloaded	Not supported	Not supported
EAR file		Automatically reloaded	Not supported	Refresh the web browser

Note: The hot method replace function automatically runs in debug mode for WebSphere Application Server.

Parent topic: [Overview: Testing and publishing on your server](#)

Related concepts:

[Hot method replace when debugging applications on WebSphere Application Server](#)

1 Tip: For WebSphere® Application Server, the hot method replace mode automatically runs in debug mode.

Servers view

Use the Servers view to manage servers. This view lists of all your servers and projects that are associated with each server. A project appears under a server when a project from the workbench is added to the server. You can use this view to start, start in debug mode, restart, or stop the servers. You can also use the Servers view to determine the status and state of a server or the projects added to the server from the workbench.

Use the Servers view for the following tasks:

- Create a server
- Edit a server
- Delete a server
- Start a server in debug mode
- Start a server
- Restart a server in debug mode
- Restart a server
- Stop a server
- Publish your application
- Restart your application **Tip:** The difference between restarting versus publishing an application on the server is that restarting an application stops and starts the application on the server; whereas, publishing an application runs the code generation tools on the application.
- Deactivate server
- Activate server
- Switch configuration
- Monitor server ports
- Add and remove projects from a server
- Run WebSphere® administration command assist, which is used for generating WebSphere administrative commands, to help develop Jython scripts.
- Run administrative console
- Run administrative script
- Reconnect debug process

Server State

The Servers view shows the current state of all the servers.

The following table lists the possible states of the server:

Table 1. Server states. Description of the server states.

Server state	Description of state
Starting	The workbench is connecting to the server and is beginning the server process. Tip: Verify that the actual server is started.
Started	The workbench is successfully connected to the server. Both the workbench and server are ready to run applications on the server.
Debugging	The workbench is successfully connected to the server. Both the workbench and server are ready to diagnose applications on the server.

Stopping	The workbench is connecting to the server and is ending the process on the server.
Stopped	The server process is complete. Or the workbench is unable to connect to the server. Tip: If you are connecting to the server and the state of the server is stopped, verify that the server is started and the port information is correctly provided in the workbench. For a WebSphere Application Server, a Test Connection link is available in the server editor to verify that you have a successful connection between the workbench and the server. Before you can use the Test Connection link, you must start the server.
Inactive	Server is not active

Server Status

The Servers view shows the status of all servers. Depending on the state of the server and the selected preferences for the server, the workbench determines the available server actions. Use the status as an indicator for possible actions.

The following table lists the possible server status:

Table 2. Server status. Description of the server status.

Server status	Description of status
Synchronized	Both the server and the applications are in sync.
Publishing	Files (projects and resource files) are copied to the correct location for the server to find and use them.
Restart	The server needs to be restarted in order for the changes to take place.
Republish	Either the server, the applications, or both are changed. Update the changed files on the server.
Restart and republish	Either the server, the applications, or both are changed. When the server is restarted, the changed files are also republished.

Project State

The Servers view might show the current state of the application. When the Servers view does not display a current state for the project, the workbench cannot retrieve the state of the project from the server or the server does not support this function.

If a project name appears in italic font style, then the project is not uploaded to the server. After the project is published into the server, the project name switches to the typical font style used in the Servers view.

The following table lists the possible states of the project: *Table 3. Project states. Description of project states.*

Project states	Description of states
----------------	-----------------------

Installing	The workbench is publishing a new application onto the server.
Starting	The application is changing from a Stopped state to a Started state. Application is starting to run but is not fully running yet. Or, it cannot fully start because one or more application modules are stopped.
Started	The application on the server is ready to run on the server.
Updating	The application is already installed on the server. File changes to the applications are being published onto the server.
Stopping	The application is changing from a Started state to a Stopped state. The application is running.
Stopped	The application on the server is ended and is not taking server requests to run.

Project Status

The Servers view might display the status of the application is either synchronized or republished. When the Servers view does not display a status for the project, the workbench cannot retrieve the status of the project from the server, the server is stopped, or the server does not support this function.

The following table lists the possible project status:

Table 4. Project status. Description of the project status.

Project status	Description of status
Synchronized	Both copies of the application files on the server and in the workbench are matching.
Republished	The application files on the workbench do not match the copy on the server. The server needs to be updated.

Parent topic: [Testing and publishing on a server](#)

Related tasks:

[Creating a server](#)

Migrating projects with a different targeted runtime into the workbench

The **targeted runtime settings** specify the application server and its version level of where you intend to run the project; for example, if the goal is to run the project on a WebSphere® Application Server V7.0, set the targeted runtime settings to WebSphere Application Server V7.0. When you specify the targeted runtime settings for a project, you can define the class path to the server. When you migrate a Java™ EE project from one workspace to another, the project may need to be set to a new target runtime because the server is no longer available. For example, if the current workbench has a WebSphere Application Server V8.0 and the project to migrate is target to WebSphere Application Server V7.0.

About this task

The class path of a project contains two class path entries. One entry is to Java developer kit and the other is to the server. The Java developer kit entry points to the directory that contains the JAR files that are necessary to support the Java developer kit version. The server entry points to the directory that contains the multiple public JAR files available in the selected server. The Java developer kit that is specified is the one required by the server. The project then compiles against the JAR files that are located in its class path. When you migrate projects with a targeted runtime environment set to WebSphere Application Server from another workspace, the Workspace Migration wizard might prompt to update the targeted runtime environment to a matching or a higher version-level of WebSphere Application Server. If a higher version-level of WebSphere Application Server is selected, the specification-level of your project resources must be compatible in the higher selected version-level of the WebSphere Application Server. For details on the WebSphere Application Server and the specification-level it supports, see the *Specifications and API documentation* topic in the WebSphere Application Server Information Center.

Restriction:

- You can migrate a project by updating the targeted runtime environment to WebSphere Application Server V7.0, V8.0 or V8.5; however, this migration tool does not support updating the targeted runtime environment to any other servers, such as Apache Tomcat server.

Procedure

1. Verify that your workspace contains a WebSphere Application Server entry in the runtime environments by completing the following steps:
 - A. In the workspace, select **Window > Preferences > Server > Runtime Environments**.
 - B. On the Server Runtime Environments page and in the **Server runtime environments** list, verify that there is a WebSphere Application Server entry. If there is no WebSphere Application Server entry, see the [Defining the server runtime environments preference](#) topic for adding a runtime environment. **Tip:** A matching or the closest higher version-level of the WebSphere Application Server specified in this Server Runtime Environment preference page is going to be the default targeted runtime environment available for selection when you bring the migrated projects into the workspace.
2. Select one of the following options to migrate your Java EE projects
 - You can use the [Existing project into Workspace](#) option.
 - You can use a source code management (SCM) system to load a project to the workbench.
 - You can permanently migrate your workspace on the same computer by starting the product with your workspace location. In this case, the settings in your Runtime Environments (**Window > Preferences > Runtime Environments**) are also preserved during the migration.
3. When the **Workspace Migration wizard** opens, this wizard helps guide you through migrating your projects or workspace to the current release of this product. In the Workspace Migration page, click **Next**.

4. In the Workspace projects which need migration page, select the projects that you want to migrate and click **Next**.
5. The Migration Project Resources page lists the workspace files that are going to be affected by the migration. If your workspace is under a source code management (SCM) system, these files are going to be checked-out from the SCM system. Click **Next**.
6. The **Undefined Server Runtime** page displays sets of projects that target a runtime environment that you must update to a server runtime environment defined in the current workbench.
 - A. For each undefined server runtime environment, under the **New Server Runtime** column, use the controls to select a server runtime environment that is defined in the current workspace.
 - B. If there is a `no supported server runtime` value displayed under the **New Server Runtime** column or you want to define a new WebSphere Application Server runtime environment for the current workspace, click the **Search for Server Runtime Environments** link to add the runtime environment to the current workbench. In the **Search for Runtime Environments** dialog box, specify the directory path to the server runtime environment, for example `C:\Program Files\IBM\WebSphere\AppServer`. Click **OK**. Under the **New Server Runtime** column, use the controls to select the newly created server runtime environment.
 - C. Select the projects that you want to change to the new server runtime environment. Click **Next**.
7. The page displays. When you are ready to begin the migration, click **Finish**.
8. When your migration is completed, the window opens. Click **OK**.
9. The Migration Results view lists the validator that were issued by the workbench during the migration. You can use this view to validate the consistency of the migrated projects in the current workspace.
10. After your project is migrated, the targeted runtime environment is updated to your selected version-level of WebSphere Application Server. To verify, go to the **Enterprise Explorer** view in the Java EE perspective, right-click your migrated project, and select **Properties > Targeted Runtimes**. Notice the **WebSphere Application Server vX** check box is selected, where `vX` is the version-level of the WebSphere Application Server.

Defining server preferences

You can define preferences using the server tools.

- **Server tools extension types**

The following table summarizes the extension types that are generated by the server tools.

- **Defining the server runtime environments preferences**

Using the Preferences page, you can direct the workbench to use a specific runtime environment of an application server for compiling, testing, or running your application.

- **Defining preferences for WebSphere Application Server**

Using the Preference page, you can define workbench options specific to WebSphere® Application Server.

Parent topic: [Testing and publishing on a server](#)

Related information:

[Defining the server preferences](#)

[Defining the audio preferences](#)

[Defining the launching preferences](#)

Server tools extension types

The following table summarizes the extension types that are generated by the server tools.

Important: Applicable edition: Full profile

Table 1. Summarizes Server tools extension types

Resource file	Extension type
servers	.server
server configurations	.config

Parent topic: [Defining server preferences](#)

Defining the server runtime environments preferences

Using the Preferences page, you can direct the workbench to use a specific runtime environment of an application server for compiling, testing, or running your application.

Procedure

1. From the **Window** menu, select **Preferences**.
2. In the Preferences window, expand the **Server** folder and then select **Runtime Environments**. The Server Runtime Environments section of the Preferences dialog is displayed.
3. Optional: Click **Search** to search your local directory for server runtime environments. Searching on a remote host is not supported.
 - A. The Search for Runtime Environments dialog box opens. In the **Folder** field, browse or type the directory where you would like the workbench to start searching for server runtime environments. Click **OK**.
 - B. If a list of installations is displayed, select the target server and click **OK**. If no installations of any application servers are found, continue completing the steps for this task.
4. Click **Add**. The New Server Runtime Environments wizard opens.
 - A. Under the **Select the type of runtime environment** list, select a server and version level as the target server runtime environment. **Tip:** In the **Select the type of runtime environment** label, you can replace the text `type filter text` with keywords to filter the list of available application servers. The following examples are filter keywords:
 - **Apache**
 - Filters the list of available servers where the vendor is Apache.
 - **WebSphere**
 - Filters the list of available servers where the name is WebSphere®.
 - **ejb**
 - Filters the list of available servers that support ejb modules.
 - **web**
 - Filters the list of available servers that support web modules.
 - **5.0**
 - Filters the list of available servers that are at version, 5.0 or supports Java™ Platform, Enterprise Edition 5 specification level.
 - **1.2**
 - Filters the list of available servers that supports Java EE 1.2 specification level or any servers that might have a version-level of 1.2.
 - B. When you add a server runtime environment, by default a server is created and added as an entry in the Servers view. If you want to add the server runtime environment and not create the server in the Servers view, clear the **Create a new local server** check box.
 - C. Click **Next**. A panel for the server is displayed.
 - D. For **Installation directory** or **Location** field, type or browse to the location of the target server. **Tip:** If you specify a false location for the target server, use the workbench to assemble modules and then use the application server to deploy the modules.

If you selected the target server as J2EE Runtime Library (a generic Java EE container), specify a directory that contains .jar files such as `\bin\lib` directory for a Java developer kit (JDK) installation.
 - E. Click **Finish**. The `target server name` and `type` variables are added to the table in the Server Runtime Environment section.

F. In the Preferences dialog, click **OK**.

What to do next

Important: If you change the target server for an EJB project at 2.1 specification-level or earlier, regenerate the deployment code by right-click the EJB project, and select **Prepare for Deployment**. If you attempt to run an application on a server target that is different from its generated deployment code, the application fails to run.

Parent topic: [Defining server preferences](#)

Related concepts:

[The WebSphere test environment](#)

Related tasks:

[Creating a WebSphere Application Server](#)

[Creating a server](#)

Related reference:

[Compilation support for WebSphere test environments](#)

Related information:

[➤ Adding the Apache Tomcat runtimes](#)

Defining preferences for WebSphere Application Server

Using the Preference page, you can define workbench options specific to WebSphere® Application Server.

About this task

Important: Applicable edition: Full profile

Procedure

1. From the menu bar of the workbench, select **Window > Preferences**.
2. In the Preference window, expand **Server > WebSphere Application Server**
3. Define the following check box:
 - **Allow applications containing errors to be published on a server**
 - Enable this check box if you want to allow applications containing errors to be published on the server. Otherwise, clear this check box if you want to prevent publishing applications that contain errors that are detected by the workbench onto a WebSphere Application Server. By default, this check box is cleared. **Tip:** You can see these errors in your application in the Problems view (**Window > Show View > Other > General > Problems > OK**).
4. In the WebSphere Application Server preference page, you can also create WebSphere Application Server profiles. See [Creating a profile for a WebSphere Application Server](#) topic for more details.

- **Profile creation for a WebSphere Application Server**

A profile is the set of files that define the runtime environment of a WebSphere Application Server.

Parent topic: [Defining server preferences](#)

Profile creation for a WebSphere Application Server

A profile is the set of files that define the runtime environment of a WebSphere® Application Server.

Important: Applicable edition: Full profile

A need for more than one profile on one server is common. For example, you might want to create separate profiles of the application server for development and testing. There are advantages of creating profiles instead of multiple installations of the WebSphere Application Server product. Not only is disk-space saved, but updating the application server is simplified when you maintain a single set of core files. Also, creating new profiles is more efficient and less prone to errors than full installations of the WebSphere Application Server.

Creating a new profile on the same server as an existing one defines unique characteristics, such as profile name and node name. Each profile shares all runtime scripts, libraries, the Java™ runtime environment, and other core server files. However, each profile has its own administrative console and administrative scripting interface.

For more information about profiles, see [Profile concept](#).

You can use the workbench in this product to start the graphical user interface tools for creating profiles. To begin, refer to the [Creating a profile in a local WebSphere Application Server V8.5, V8.0 or V7.0](#) topic

Parent topic: [Defining preferences for WebSphere Application Server](#)

Parent topic: [Creating, editing, and deleting servers](#)

Parent topic: [Configuring a WebSphere Application Server](#)

Creating, editing, and deleting servers

You can create a server to identify the runtime environment that you want to use for testing your project resources. When creating a server, you also create a pointer from the workbench to an existing installation of an application server.

- [Creating a server](#)

You can create a server to identify the runtime environment that you want to use for testing your project resources. The term creating a server defines creating a pointer from the workbench to an existing installation of an application server.

- [Profile creation for a WebSphere Application Server](#)

A profile is the set of files that define the runtime environment of a WebSphere® Application Server.

- [Creating a profile in a local WebSphere Application Server V8.5, V8.0 or V7.0](#)

Parent topic: [Testing and publishing on a server](#)

Related information:

[Editing a server](#)

[Deleting a server](#)



<http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.wst.server.ui.doc.user%2Ftopics%2Ftcreate.html&resultof=%22creating%22%20%22creat%22%20%22editing%22%20%22edit%22%20%22servers%22%20%22server%22>

Creating a server

You can create a server to identify the runtime environment that you want to use for testing your project resources. The term creating a server defines creating a pointer from the workbench to an existing installation of an application server.

About this task

Procedure

1. Select one of the following options:
 - Select **Window > Show View > Servers**. Right-click **New > Server**.
 - In the menu bar, click **File > New > Other**. Expand the Server folder, select **Server** and click **Next**.
 2. The New Server wizard opens. This wizard defines a new server. The new server contains information that you can use to:
 - Point to a specific runtime environment for local or remote testing.
 - Publish to an application server.
 3. In the **Select the server type** list, select the type of server or test environment where you want to publish or test your resources.
 - A. In the text field under the **Select the server type** label, you can replace the text `type filter text` with keywords to filter the list of available application servers. The following are examples of filter keywords:
 - **Apache**
 - Filters the list of available servers where the vendor is Apache.
 - **WebSphere**
 - Filters the list of available servers where the name is WebSphere®.
 - **ejb**
 - Filters the list of available servers that support ejb modules.
 - **web**
 - Filters the list of available servers that support web modules.
 - **5.0**
 - Filters the list of available servers that are at version, 5.0 or supports Java™ Platform, Enterprise Edition 5 specification level.
 - **1.2**
 - Filters the list of available servers that supports Java EE 1.2 specification level or any servers that have a version-level of 1.2.
 - B. There is a brief description about the server that is selected from the server type list.
 4. In the **Server's host name** field, you can provide the fully qualified DNS name or IP address of the host machine where the server is running. The default address is: `localhost`.
 5. The **Server runtime environment** list is available when you select a server from the server type list that contains a runtime environment entry in the Runtime Environments preferences page **Window > Preferences > Server > Runtime Environments**. If the **Server runtime environment** list is not available, continue to the next page of the New Server wizard. The wizard prompts for the installation directory of the server. After you complete this New Server wizard, the workbench adds a runtime environment entry in the Runtime Environments preference page.
 - To compile, test, or run your application, select the **Server runtime environment** list. Select the runtime environment of an application server.
 - To modify the server runtime environment, select **Configure runtime environments**.
 - To add a new server runtime environment, select **Add**.
-

- For more information about runtime environments, see Defining the server runtime environments preferences.
6. Click **Next**. Follow the instructions in the wizard to specify the details of the server that you want to create.
 7. After completing the New Server wizard, click **Finish**. You can view the new server in the Servers view.

Related tasks:

[Defining the server runtime environments preferences](#)

[Creating a WebSphere Application Server](#)

[Testing on a Web Preview Server](#)

[Testing on a Java Platform, Enterprise Edition Preview server](#)

Related reference:

[Servers view](#)

Related information:

[Testing on an HTTP Preview server](#)

[Creating a Tomcat server](#)

Creating a WebSphere Application Server

You can create a server to specify a WebSphere® Application Server runtime environment for testing or publishing your project resources. Example usages of a runtime environment are compiling an application, connecting to a server, and publishing applications on a server.

About this task

The task of creating a WebSphere Application Server involves creating a reference from the workbench to an existing installation of an application server and its profile. You can use this reference to handle your server requests from the workbench.

Important: Applicable edition: Full profile

Prerequisite

- An installation of WebSphere Application Server installed either on the same or on a remote computer from the development workbench

If you want to create a reference to a local installation of the WebSphere Application Server or you want to create a reference to a remote installation of the WebSphere Application Server, complete the following steps.

Procedure

1. Select one of the following options:
 - Select **Window > Show View > Servers**. Right-click **New > Server**. In the menu bar, click **File > New > Other**. Expand the **Server** folder, select **Server** and click **Next**.
 - The New Server wizard opens. This wizard defines a new server. The new server contains information that you can use to:
 - Point to a specific runtime environment for local or remote testing.
 - Publish to an application server.For more information, see [Creating a server](#).
2. Select any of the following options:
 - In the Servers view (**Window > Show View > Servers**), right-click and select **New > Server**.
 - In the menu bar, click **File > New > Other**. Expand the **Server** folder, select **Server**, and click **Next**.
3. The New Server wizard opens. This wizard defines a new server, that contains information required to point to a specific runtime environment for local or remote testing, or for publishing to an application server.
4. In the **Select the server type** list, under the IBM® folder, select one of the following servers you want to create:
 - **WebSphere Application Server v8.5**
 - **WebSphere Application Server v8.0**
 - **WebSphere Application Server v7.0**
5. In the **Server's host name** field, you can provide the fully qualified domain name system (DNS) name or Internet protocol (IP) address of the host machine that where the server is running. By default, this field is completed with the default address: `localhost`. This localhost address is used for connecting to a local server. If you want to connect to a remote server, specify its DNS or IP address in this field.
6. In the **Server name** field, type a label to identify this server entry in the Servers view. By default, this field is completed with the following naming conventions; *server type at host name*, for example `WebSphere Application Server v8.5 at localhost`.
7. If the workbench has a reference to a WebSphere Application Server runtime environment, a **Server runtime environment** list is available to select a runtime environment. When you select a runtime environment, follow these guidelines:
 - If you want to perform actions on a local installation of the WebSphere Application Server, you must select the runtime

environment that defines an installation directory to a local server. When you publish the application on the server, the runtime environment publishes the application in the directories of the local server.

- If you want to perform actions on a remote installation of the WebSphere Application Server, select a local installation of the server. When you are publishing the application on the server, the application does not publish in the local installation of the server or the stub directory; instead, these runtime environments contain libraries to connect and publish to the remote server.

If you want to add runtime environments, click **Add**. If you want to modify the runtime environments that are defined in the workbench, click **Configure runtime environments**.

8. **Note:** Step 7 is only available in the server editor.

Click **Next** to configure more settings.

9. If the workbench does not have a reference to the server runtime environment, the New Server wizard prompts for this information in the **WebSphere Application Server Runtime Environment** page.

A. In the **Name** field, type a label to identify this server entry in the Server Runtime Environments preference page (**Window > Preferences > Server > Runtime Environments**).

B. In the **Installation directory** field, type or browse to the path where the WebSphere Application Server is installed. This path is the same as the WAS_ROOT path mappings as defined by the WebSphere Application Server configuration. For example, if you have installed WebSphere Application Server in C:\WebSphere\AppServer directory, then specify this path in the **Installation directory** field.

C. Click **Next** to configure WebSphere Application Server settings.

10. On the **WebSphere Application Server Settings** page, specify one or more of the following settings.

Option	Description
Profile name	Select the name of the profile of WebSphere Application Server. A profile is the set of files that define the runtime environment. For information on creating a profile, see Creating a profile . This option is available only if you are running a local server and not available if you are running a remote server.

Server connection type and administrative port

Administrative ports are used to communicate requests between the workbench and the server. Use one of the following radio buttons to select how you want the workbench to connect to the server: The **Automatically determine connection settings** option is available only for local servers and by default is selected when you are working with a local server. It retrieves the port values directly from the configuration files of the server that is defined in its profile and attempts to connect to one of these available ports. Use the **Manually provide connection settings** option to select connection types. This option is available for both local and remote servers. For remote servers, you are restricted to using this option and you must provide the correct port numbers for each connection type you choose. The following ports are used for making Java™ Management Extensions (JMX) connections with the server: The InterProcess Connector (IPC) port is a more stable and robust connection between the workbench and the local server. The default setting of the IPC port is 9633 The remote method invocation (RMI), also known as the ORB bootstrap, port is designed to improve performance and communication with the server. The default setting of the RMI port is 2809 The SOAP connector port is firewall compatible. The default setting of the SOAP port is 8880 For more information, see [Setting the connection to the WebSphere Application Server](#).

<p>Run server with resources within the workspace</p>	<p>Use this check box enables to run your application from within the workbench without publishing to the server. As a result, your application is not copied to the server under the installedApp folder <i>(x: /profiles/profile_name/installedApps, where x is the installation directory where WebSphere Application Server is installed)</i>. This option is available only if you are running a local server and not available if you are running a remote server. For local servers, this check box is enabled by default. If this check box is clear, the workbench runs the server with resource on the server. Your application publishes on the targeted server. The enterprise application (EAR) file is automatically published to the server by expanding into the installedApps folder <i>(x: /profiles/profile_name/installedApps, where x is the installation directory where WebSphere Application Server is installed)</i></p>
<p>Enable remote server start</p>	<p>Starts the WebSphere Application Server from the workbench. For more information about enabling to start a remote WebSphere Application Server, see Starting a remote WebSphere Application Server</p>
<p>Security is enabled on this server</p>	<p>Enables the security feature that comes with WebSphere Application Server. When security is not enabled, all other security settings are ignored. For more information about this security option, see Specifying administrative settings to a secured WebSphere Application Server.</p>
<p>Application server name</p>	<p>Specifies a logical name for the application server. For WebSphere Application Server, the logical name is unique and assigned to a server that distinguishes it from all other server instances within the node. This server name must already be created in the application server and its default setting is <i>server1</i>.</p>

11. Click the **Test Connection** link to verify that you have a successful connection between the workbench and the server.

Before you can test connection, you must start the server. **Tip:** To ensure a successful connection, verify the following prerequisites:

- The actual server is started.
- A profile to the server runtime environment is correctly specified under the **Profile name** field.
- If you are manually providing the connection settings, you must specify the correct port values.

12. If you are creating a reference to a remote server and **enable remote server start** is selected, the Remote WebSphere Application Server Settings page opens.
 - Select whether your remote server is installed on Windows or Linux, and enter the location of the server profile.
 - If you want to access the remote server with logon credentials, enter your user name and password.
 - If you want to access the remote server with SSH, copy the private key file to the machine where the workbench is installed, and specify the key file location and user ID.For more information about starting a remote WebSphere Application Server, see [Starting a remote WebSphere Application Server](#) topic.
13. Click **Next** to add the projects of your application to the server. On the **Add and Remove Projects** page, under the **Available projects** list, select the project that you want to test and click **Add**. The project appears in the configured projects list.
14. If you have the option to click **Next** and you select it, the **Select Task** page opens. In the Select Task page, use the check boxes to select tasks to be performed on the server, such as create tables and data source.
15. Click **Finish**.

Related concepts:

[The WebSphere test environment](#)

Related tasks:

[Publishing settings for a WebSphere Application Server](#)

[Defining the server runtime environments preferences](#)

[Creating a server](#)

Related reference:

[Compilation support for WebSphere test environments](#)

[Servers view](#)

[Unable to connect to a remote WebSphere Application Server hosted on a Red Hat Enterprise Linux machine](#)

Testing on a Java Platform, Enterprise Edition Preview server

Use the Java™ Platform, Enterprise Edition Preview server when you want to quickly compile, test, and run resources in a Java EE web project. The Java EE web projects that you can test on a Java Platform, Enterprise Edition Preview server are static web projects, dynamic web projects, and utility projects. The term Java EE web project refers to any of these projects.

About this task

The Java Platform, Enterprise Edition Preview server is an embedded application server that is integrated with the workbench. It is intended for preliminary testing and saves you the time to download, install, and set up an application server. However, as your Java EE web project progresses through its development cycle you are encouraged to test and run on an external application server. You can use the Java Platform, Enterprise Edition Preview server as a convenient way to check for errors before you do a final test on an external application server. To create and test on a Java Platform, Enterprise Edition Preview server for previewing your resources in a Java EE web project, complete the following steps.

Procedure

1. In the Servers view, right-click and select **New > Server**. The Define a New Server wizard opens.
2. Under the **Select the server type** list, expand the **Basic** folder and select **J2EE Preview**. Click **Next**.
3. The **Add and Remove** page opens. Under the Available projects list, select the Java EE web project and click the **Add** button. Click **Finish**. The Java Platform, Enterprise Edition Preview server is created in the Servers view.
4. In the Enterprise (or Project) Explorer view, navigate to your Java EE web project and right-click your resource that you want to test.
5. Select **Run As > Run on Server**. The **Run on Server** wizard opens.
6. Verify the **Choose an existing server** radio button is enabled.
7. Under the **Select the server that you want to use** list, select the recently created Java Platform, Enterprise Edition Preview server. Click **Next**.
8. The **Add and Remove** page opens. Verify the Java EE web project that you are testing is listed under the **Configured projects**. Click **Finish**.

Results

The internal web browser opens and displays the web content.

Parent topic: [Testing applications on a server](#)

Related tasks:

[Creating a server](#)

Creating a profile on a local WebSphere Application Server V8.5, V8.0, or V7.0

How to create runtime environments for a local WebSphere® Application Server V8.5, V8.0, or V7.0. Each runtime environment is created within a profile. A profile is the set of files that define the runtime environment.

Before you begin

The Profile Management tool is a graphical user interface to the WebSphere Application Server command line tool, `manageprofiles` for creating profiles. You can use the development workbench in this product to start the Profile Management tool that is provided by the WebSphere Application Server product.

You can use the Profile Management tool to create multiple profiles on the same server. The need for defining multiple profiles to create multiple runtime environments on the same server is common. An example use case is allowing different teams to test independently of each other on the same server by using their own profiles. Another example use case is allowing a developer to create separate profiles of the application server for development and testing.

Prerequisites:

- See the [Profiles: required disk space](#) topic from the WebSphere Application Server information center to verify that you have the required disk space.
- Set the user of your operating system to have administrator authority, which is the default setting to run the Profile Management tool for WebSphere Application Server. For example, the root user on Linux. For details to create a profile with a non-root user, see the *Managing profiles for non-root users* topic available in the WebSphere Application Server information Center:
 - For WebSphere Application Server V8.5, see the [Managing profiles for non-root users](#) link
 - For WebSphere Application Server V8.0, see the [Managing profiles for non-root users](#) link
 - For WebSphere Application Server V7.0, see the [Managing profiles for non-root users](#) link
- **Note:** Concurrent profile creation is not supported for one set of core WebSphere Application Server product files. Concurrent attempts to create profiles result in a warning about a profile creation already in progress.

About this task


Important: Applicable edition: Full profile

To create a profile on a local WebSphere Application Server:

Procedure

1. From the menu bar of the workbench, select **Window > Preferences**.
2. In the Preferences window, expand **Server > WebSphere Application Server**.
3. In the **WebSphere Application Server local profile management** list, select a server runtime environment for which you want to create a profile. **Tip:** To add a server runtime environment to this list, see the [Defining the server runtime environments preferences](#) topic.
4. Click **Run Profile Management Tool**.
5. Configure your profile by following the instructions in the Profile Management tool. For more information about configuring a profile by using the Profile Management tool, see [Using the Profile to create an application server](#). **Tip:** **Windows** If you select the Advanced profile creation option, in the Windows Service Definition page of the Profile Management tool, deselect the **Run the Application Server process as a Windows service** check box. Window services are global settings on an operating system. Deselecting this check box avoids conflict of service requests between different profiles. For example, if you configure an application server as a Windows service and issue the

startServer command, the **wasservice** command attempts to start the defined service. As a result, you can lose track of which profile issued the startServer command, as any profile can start this globally defined Window service.

6.  In order for the workbench to communicate to the server, you must grant a non-root user access to the newly created profile. See the [Creating profiles for non-root users](#) topic for details.

7. Optional: You can delete a profile:

A. In the **WebSphere Application Server local profile management** list, select the server runtime environment that contains the profile that you want to delete.

B. In the **WebSphere Application Server profiles defined in the runtime selected above** list, select the profile for which you want to delete and click **Delete**.

C. A dialog box displays with the following text: `WebSphere Server Warning`

```
Deleting the profile will remove the files that are associated with this profile. Do you want to continue with this operation?
```

Click **Yes** to confirm if you want to remove from the file system the registry and configuration files that are associated to the profile. Otherwise, click **No** to cancel the process of deleting the profile.

D. If you clicked **Yes**, after the registry and configuration files of the profile are deleted, another dialog box appears with the following text: `WebSphere Server Warning`

```
The profile directory contains files you have created or modified. Do you want to delete these files?
```

Click **Yes** to confirm if you want to remove from the file system the remaining log files that are associated to the deleted profile. Otherwise, click **No** to keep the remaining log files in the file system.

Parent topic: [Creating, editing, and deleting servers](#)

Related reference:

[Profile creation for a WebSphere Application Server](#)

Configuring servers for testing and publishing

A server is a definition that identifies where an application is going to be tested or published and points to a specific runtime environment, such as a WebSphere® Application Server on a local server or on another server. Select the server that you want to configure:

About this task

Important: Applicable edition: Full profile

- [Configuring a WebSphere Application Server](#)

A server is a definition that identifies where an application is going to be tested or published and points to a specific runtime environment such as a WebSphere Application Server on a local server or on another server. The following topics describe how to set up a WebSphere Application Server, such as setting up ports or data sources.

- [Configuring a web Preview Server](#)

You can configure your web Preview Server and the `proxy-config.xml` file.

Parent topic: [Testing and publishing on a server](#)

Related reference:

[Advanced WebSphere Application Server configurations](#)

Configuring a WebSphere Application Server

A server is a definition that identifies where an application is going to be tested or published and points to a specific runtime environment such as a WebSphere® Application Server on a local server or on another server. The following topics describe how to set up a WebSphere Application Server, such as setting up ports or data sources.

- [Profile creation for a WebSphere Application Server](#)

A profile is the set of files that define the runtime environment of a WebSphere Application Server.

- [Setting status updates for a WebSphere Application Server](#)

You can specify how long the development environment is going to wait to update the status of the server that is running in real time.

- [Setting the connection to the WebSphere Application Server](#)

The administrative ports are used to communicate requests between the workbench and the server.

- [Optimizing starting the WebSphere Application Server for development](#)

There are two options available to help reduce the start time of the WebSphere Application Server: **Start server with a generated script** and **Run -Xquickstart in the JVM settings if applicable**. These options are available as check box options in the workbench.

- [Switching to HPEL mode for logging and tracing on WebSphere Application Server V8.0 or later](#)

There are two modes of logging and tracing for WebSphere Application Server V8.0 or later, which are basic mode and High Performance Extensible Logging (HPEL) mode. HPEL is designed to provide faster log and trace handling capabilities than the basic mode.

- [Keeping the WebSphere Application Server running after exiting the development environment](#)

To specify that the server will continue running after you shut down the development environment.

- [Changing the hypertext transfer protocol settings for WebSphere Application Server](#)

Hypertext transfer protocol (HTTP) is an Internet Protocol that is used to transfer and display hypertext and XML documents on the web. Hypertext transfer protocol secure (HTTPS) is an encryption and authentication layer added to HTTP, which is used by web servers and web browsers to transfer and display hypermedia documents securely across the Internet.

- [Publishing settings for a WebSphere Application Server](#)

Publishing involves copying files (application, resource files, and deployment descriptor files) to the correct location for the server to find and use them. You can either publish your application on the server or run your application within the development environment without copying the application into the directories of the server.

- [Specifying administrative settings to a secured WebSphere Application Server](#)

If your WebSphere Application Server runtime environment has global security that is enabled, you need to communicate the administrative settings from your development environment to the runtime server. On the workbench, you need to specify that security is enabled on the runtime environment, and provide the user name and password to the secured server. In addition, you need to establish a trust between the development workbench of this product and the server.

- [Starting a remote WebSphere Application Server](#)

Use the workbench to start a remote WebSphere Application Server running on a Windows or Linux operating system. You can choose to use an operating system or Secure Shell (SSH) authentication method to access the remote host that is running the server. After you provide the workbench with authentication details to access the remote host, you can start the remote server in start or debug mode.

- [Configuring a WebSphere Application Server with Federal Information Processing Standard enabled](#)

You can set up the workbench to connect to a WebSphere Application Server with Federal Information Processing Standard (FIPS) enabled.

Parent topic: [Configuring servers for testing and publishing](#)

Setting status updates for a WebSphere Application Server

You can specify how long the development environment is going to wait to update the status of the server that is running in real time.

About this task

Important: Applicable edition: Full profile

To configure the status updates for a WebSphere® Application Server:

Procedure

1. In the Servers view, double-click your WebSphere Application Server to open the server editor.
2. Click the **Overview** tab.
3. Expand the **Server** section.
4. In the **Update server status interval (in milliseconds)** field, specify how long the development environment waits before you update the **Status** column in the Servers view with the status of the server that is running in real time.
5. Save and close the editor.

Parent topic: [Configuring a WebSphere Application Server](#)

Setting the connection to the WebSphere Application Server

The administrative ports are used to communicate requests between the workbench and the server.

About this task

Important: Applicable edition: Full profile

The following ports are used for making Java™ Management Extensions (JMX) connections with the server:

- The remote method invocation (RMI), also known as the ORB bootstrap, port is designed to improve performance and communication with the server. The RMI connection is JSR 160 RMI compliant. The default setting of the RMI port is 2809
- The SOAP connector port is more firewall compatible. The default setting of the SOAP port is 8880
- The InterProcess Connector (IPC) port is a more stable and robust connection between the workbench and the local server. The default setting of the IPC port is 9633

If your operating system is configured with Federal Desktop Core Configuration (FDCC) and you want to connect to a WebSphere® Application Server on another server as your workbench, you need to use the SOAP connector port to cross the firewall that is typically present between the workbench and the remote application server.

To configure the server connection type and port number for WebSphere Application Server:

Procedure

1. In the Servers view, double-click the WebSphere Application Server you want to modify. The server editor opens.
2. Click the **Overview** tab.
3. Expand the **Server** section.
4. Under the **Server connection types and administrative ports**, use one of the following radio buttons to select how you want the workbench to connect to the server:
 - The **Automatically determine connection settings** option is available only for local servers and by default is selected when you are working with a local server. It retrieves the port values directly from the configuration files of the server that is defined in its profile and attempts to connect to one of these available ports.
 - The **Manually provide connection settings** option allows you to select which connection types you want to use. It is available for both local and remote servers. If you are using a remote server, you are restricted to using this option and you must provide the correct port numbers for each connection type you choose.
5. In the case you selected the **Manually provide connection settings** the radio button, complete the table:
 - A. Under the Connection Type column, you can select more than one connection type for communicating between the development environment and the server. The workbench determines the preferred ordering for which connection type to use. This decision depends on factors such as the version-level of the server and whether the server is local or remote. If the workbench loses a connection during a session, the workbench is going to switch connection types and attempt to establish another connection to the server.
 - B. Under the Port column, if the server is remote, specify the port value for each of the connection type you selected in the previous step. If the server is local, the port values are automatically retrieved from the configuration files of the server that is defined in its profile. **Tip:** The default port value that is assigned to each connection type is referenced under the default port column. To determine the actual port values of a configured profile reference its server configuration files. The port values are stored in the `serverindex.xml` file that is located in the following directory: `x:\profiles\<profileName>\config\cells\<cellName>\nodes\<nodeName>`, where *x* is the directory WebSphere Application Server is installed.
6. Optional: Click the **Test Connection** link to verify if you have a successful connection between the workbench and the server. Before you use the Test Connection link, the server requires to be started. **Tip:** To ensure a successful

connection, verify that:

- The actual server is started.
- A profile to the server runtime environment is correctly specified under the **Profile name** field.
- If you are manually providing the connection settings, verify that the correct port values are specified.

7. Save and close the editor. When you save the modifications to the server editor, the server tools attempts to make a connection to the server using your latest saved changes.

Parent topic: [Configuring a WebSphere Application Server](#)

Related tasks:

[Creating a WebSphere Application Server](#)

Optimizing starting the WebSphere Application Server for development

There are two options available to help reduce the start time of the WebSphere® Application Server: **Start server with a generated script** and **Run -Xquickstart in the JVM settings if applicable**. These options are available as check box options in the workbench.

About this task

These options can start the application server quickly for testing and development purposes. However, for production servers, disable these options where long run throughput is more important than early start of the server.

Important: Applicable edition: Full profile

- Start server with a generated script

- This option uses the `startServer -script` command that is provided by the WebSphere Application Server to generate a script. It starts the server with this previously generated script instead of using the `startServer` command directly. For more details about the `startServer` command, see the [startServer command](#) topic available in the information Center for WebSphere Application Server.

- Run -Xquickstart in the JVM settings if applicable

- This option uses the Java™ virtual machine (JVM) setting `-Xquickstart` on startup when the server starts in run or profile mode. However, the `-Xquickstart` JVM setting is ignored when the server starts in debug mode or runs on the Solaris operating system. For details on the `-Xquickstart` property for the JVM setting, see [Java virtual machine settings](#) topic. **Tip:** Although this check box is clear from the workbench, the `-Xquickstart` property might remain set in the JVM by another server setting. You can try to clear the `-Xquickstart` setting from the JVM by also disabling the **Run in development mode** check box from the administrative console of the WebSphere Application Server. For details on the **Run in development mode** check box, see the [Application server settings](#) topic.

The default setting for these options depend on your server configuration.

- If you create a local server configuration, by default these check boxes are enabled.
- If you create a remote server configuration, by default these check boxes are disabled.
- If you have a server configuration that is migrated from a previous version of this product, by default these check boxes are disabled.

To configure the **Optimize server start for development** settings:

Procedure

1. In the Servers view, double-click your WebSphere Application Server to open the server editor.
2. On the **Overview** page under the **Server** section, there are two check boxes available under the **Optimize server start for development** list.
 - Select the **Start server with a generated script** check box.
 - Select the **Run -Xquickstart in the JVM settings if applicable** checkbox.
3. Type **Ctrl+S** to save the server configuration.
4. Close the editor.
5. Start the server:
 - A. In the Servers view, right-click the server.
 - B. Select **Start**.

Parent topic: [Configuring a WebSphere Application Server](#)

Related tasks:

[Starting a server](#)

Switching to HPEL mode for logging and tracing on WebSphere Application Server V8.0 or later

There are two modes of logging and tracing for WebSphere® Application Server V8.0 or later, which are basic mode and High Performance Extensible Logging (HPEL) mode. HPEL is designed to provide faster log and trace handling capabilities than the basic mode.

Before you begin

Viewing log and traces in HPEL mode using the workbench is only available for WebSphere Application Server V8.0 or later running on Windows or Linux operating systems.

Important: Applicable edition: Full profile

About this task

The **basic** mode is the default logging and tracing setting for the server. In basic mode, the log and trace contents are written in plain text format to the log files, such as `SystemOut.log`, `SystemErr.log`, `trace.log`, and `activity.log` files. In **HPEL** mode, the log and trace contents are written to a log data or trace data repository in a proprietary binary format. The use of a binary log format instead of a plain text format is designed to improve performance of the server by providing faster log and trace handling capabilities. To enable HPEL mode, you must use the administrative console or `wsadmin` scripting that is provided by WebSphere Application Server. You can use the workbench to translate a snapshot of the binary (HPEL) server logs in to readable text that displays in the editor pane. In addition, during the run time of the server you can view HPEL log and traces in the Console view in a readable text format.

The log and trace records in HPEL mode uses the same message format as the basic mode. The following table describes the data that makes up a log record:

Data	Description
Time Stamp	The time when the event was recorded.
Thread ID	The identity of the thread that recorded the event in hexadecimal notation.
Logger	The logger that recorded the event.
Level	The type of event that was recorded.
Message	The message from the recorded event. If the message has a message ID, the message ID is underlined. In the Console view, you can click the message ID to get an explanation and recommended user action for the message. In the editor pane, the message ID link is not available and display as plain text only.

Here is an example of a log record: `[3/3/11 23:01:30:147 EST] 0000000f ApplicationMg Z WSVR0221I: Application started: DefaultApplication`

In this example, `[3/3/11 23:01:30:147 EST]` is the time stamp, `0000000f` is the thread ID, `ApplicationMg` is the logger, `z` is the level, and `WSVR0221I: Application started: DefaultApplication` is the message where `WSVR0221I` is the message ID.

Procedure

1. In the Servers view, right-click the **WebSphere Application Server V8.0** server entry and select **Properties > WebSphere Application Server**.

2. Under the **Server log directory** section, the default settings of basic mode show **Open SystemOut.log in editor** and **Open SystemErr.log in editor** links. You can click these links to display the basic mode of the log and trace content of the `SystemOut.log` and `SystemErr.log` files in the editor pane.
3. You must use the administrative console or `wsadmin` scripting tools that are provided by WebSphere Application Server to switch from the basic mode to HPEL mode by completing the [Changing from basic mode to HPEL logging and tracing](#) topic in the information Center for WebSphere Application Server. **Tip:** You can open the administrative console within the workbench by going into the Servers view, right-click the server and select **Administration > Run Administrative Console**. For more information, see [Accessing the administrative console](#).
4. Restart the server to enable the server configuration changes. In the Servers view, right-click the **WebSphere Application Server V8.0** server entry and select **Restart**.
5. In the Servers view, right-click the **WebSphere Application Server V8.0** server entry and select **Properties > WebSphere Application Server**.
6. When the server is in HPEL mode, under the **Server log directory** section shows the **Display a snapshot of the binary server logs** link. You can click this link to display in the editor pane a capture at a point in time of the log and trace content in HPEL mode; which the workbench translates from binary and displays as readable text. The snapshot of the binary (HPEL) server log includes both `System.out` and `System.err` output and displays as a single file in the editor pane. **Tip:** If you want an update of the logs and traces as the server continues running in HPEL mode, you must click **Display a snapshot of the binary server logs** link again, to generate another snapshot report of the binary (HPEL) log file.

What to do next

- You can use the administrative console to view and filter through the HPEL log and trace content. For more information, see [Log viewer settings](#).
- You can use the administrative console or `wsadmin` scripting tools to switch back to the basic mode from HPEL mode. For more information, see [Changing from HPEL to basic mode logging and tracing](#).

Parent topic: [Configuring a WebSphere Application Server](#)

Keeping the WebSphere Application Server running after exiting the development environment

To specify that the server will continue running after you shut down the development environment.

About this task

Important: Applicable edition: Full profile

Procedure

1. In the Servers view, double-click the server. The server editor opens.
2. On the Overview page, under the Server section, clear the **Terminate server upon workbench shutdown** check box.
By default, this check box is clear.
3. Save and close the editor.
4. After the development environment is restarted, the workbench automatically reconnects to the server when you open the Servers view.
 - A. If the server is connected successfully, the status of the server in the Servers view displays any of the following status.
 - **Started:** If the remote server was initially started by right-clicking the server and select **Start**.
 - **Debugging:** If the remote server was initially started by right-clicking the server and select **Debug**.
 - **Profiling:** If the remote server was initially started by right-clicking the server and select **Profile**.

Parent topic: [Configuring a WebSphere Application Server](#)

Parent topic: [Managing servers](#)

Changing the hypertext transfer protocol settings for WebSphere Application Server

Hypertext transfer protocol (HTTP) is an Internet Protocol that is used to transfer and display hypertext and XML documents on the web. Hypertext transfer protocol secure (HTTPS) is an encryption and authentication layer added to HTTP, which is used by web servers and web browsers to transfer and display hypermedia documents securely across the Internet.

Before you begin

- In the Servers view, you must have a [WebSphere® Application Server](#) defined.

Important: Applicable edition: Full profile

About this task

You can control the hypertext transfer protocol (HTTP) settings of a WebSphere Application to run in either HTTP or HTTPS. By default, the workbench uses HTTP to communicate between the web browser and server. You can change the workbench to use HTTPS. In the workbench, the following actions can use HTTPS:

- [Run on Server](#)
- [Run Administrative Console](#)

If the workbench is set to use HTTP and you want to run the administrative console, the workbench starts the web browser with HTTP. However, the WebSphere Application Server automatically redirects to use HTTPS when the administrative security is enabled on the server.

To change the hypertext transfer protocol settings for WebSphere Application Server:

Procedure

1. In the Servers view, right-click the WebSphere Application Server and select **Open**.
2. Under the **Server** section in the **Overview** page of the server editor, there is a **Use HTTPS when running server resources** check box. Select this check box to enable HTTPS, or clear this check box to enable HTTP. By default, this check box is clear.
3. Save and close the editor to activate your changes.

Parent topic: [Configuring a WebSphere Application Server](#)

Related reference:

[The internal browser cannot display the administrative console for a secured WebSphere Application Server](#)

Publishing settings for a WebSphere Application Server

Publishing involves copying files (application, resource files, and deployment descriptor files) to the correct location for the server to find and use them. You can either publish your application on the server or run your application within the development environment without copying the application into the directories of the server.

About this task

Note: You can use the workbench to publish or remove applications to any WebSphere® Application Server that it supports. Although it is possible to deploy applications into production directly from your development environment, this is not a recommended practice. Access control should be enforced on production servers, and applications should deploy through controlled and repeatable processes.

Important: Applicable edition: Full profile

Run server with resources on Server

The **run server with resources on Server** publishing option copies the full application and its server-specific configuration from the workbench into the directories of the server. To use this publishing option, the server can be either a remote or local WebSphere Application Server. The default location where an application gets copied into the server is `\directory\profile\installedApps\cellName` directory, where `\directory\profile` is the directory of your profile for the WebSphere Application Server.

The advantage of selecting the **run server with resources on Server** setting is you are running your application from the directories of your server and you can edit advanced application-level settings by using the administrative console.

However, this publishing option might take a longer time to complete than the **run server with resources within the workspace** publishing option, as it involves more files that are copied to the server.

If you want to switch publishing settings to **run server with resources within the workspace**, you need to remove the application from the server by using the **Add and Remove Projects** wizard, select the **run server with resources within the workspace** radio button, and add the application back onto the server by using the **Add and Remove Projects** wizard.

Run server with resources within the workspaceThe **Run server with resources within the workspace** publishing option requests the server to run your application from the workspace. This publishing option is available only when you are running a local WebSphere Application Server and not available when you are running a remote server.

The **run server with resources within the workspace** setting is useful when you are developing and testing your application as it is designed to operate faster than the **run server with resources on Server** publishing option as fewer files are involved when copied over to the server.

This publishing option should publish faster when an application contains a single root, as opposed to containing multiple roots, because the server expects the structure of an application to have only a single root. As a result, the workbench might require more processing time to publish an application with multiple roots. To determine whether the structure of your application contains a single or multiple roots, use the Project Structure Validator. For details, see [Creating and configuring Java™ EE projects using wizards](#) topic.

If you enable the **run server with resources within the workspace** setting and clear the **Minimize application files copied to the server** publishing option, when you select to add your application to the server by using the **Add and Remove Projects** wizard, the application does not get copied into the directory of the server. For example, the application files do not get copied into the `installedApps` directory of the server. However, your application does get copied into your server configuration directory: `\directory\profile\config\cells\cellName\applications`, where `\directory\profile\` is the directory of your profile for the WebSphere Application Server. **Important:** When you are using the **Run server with resources within the workspace** option, you can view only your deployment descriptor file by using the administrative console. In addition, you cannot edit application-level configurations by using the administrative console, including Java EE configurations, enhanced EAR settings, policy set attachments, bindings, and other settings. For example, an

enhanced EAR setting that is not available for editing in the administrative console is the class-loader option `PARENT_LAST`. You are limited to editing your application-level configurations in the workbench.

If you want to switch publishing settings to **run server with resources on Server**, you need to remove the application from the server by using the **Add and Remove Projects** wizard, select the **run server with resources on Server** radio button, and add the application back onto the server by using the **Add and Remove Projects** wizard.

CAUTION:

When you are using "Run server with resources within the workspace" publishing option, the server can lose track of your application under the following scenarios:

- **If you delete your workspace, the server no longer can find your application. As a result, if you did not put your application under source control management and the workspace is deleted, you can lose your application from your file system.**
- **If you delete an application from the workspace without removing it from the server, the server no longer can find your application. As a result, you might encounter errors when you are starting the server because the server tries to start the missing application from the workspace. You can try to manually remove the remaining application files from the server by using the administrative console or the `wsadmin` command-line tool.**

Minimize application files copied to the serverAn extra publishing option becomes available when the **run server with resources within the workspace** option is selected, which is the **Minimize application files copied to the server** option. This publishing option is available only when you are running a local WebSphere Application Server and the application runs from the workspace, and not available when you are running a remote server. It is designed to optimize the publishing-time on the server by reducing the files that are copied to the server. In addition to the application files not getting copied into the `installedApps` directory of the server, the application also does not get copied into your server configuration directory. **Important:** The same restrictions for viewing and editing application-level configurations in the administrative console that is described in the [Important](#) section under the **Run server with resources within the workspace** option applies to this publishing option. An extra restriction for the **Minimize application files copied to the server** publishing option is that some configuration options appear missing in the administrative console, which is designed to prevent modifications by using the console and instead to use the workbench for editing the application-level configurations.

If you want to be able to view your application's deployment descriptors in the **administrative console**, you need to remove the application from the server by using the **Add and Remove Projects** wizard, disable the **Minimize application files copied to the server** publishing setting by clearing this check box and enable only the **run server with resources on Server** radio button, and then add the application back onto the server by using the **Add and Remove Projects** wizard. **Automatically start applications after publishing**Select the **Automatically start applications after publishing** check box if you want the workbench to make an automatic attempt to start the application after it is published to the server. By default this check box is enabled.

You can clear this check box if you want to manually start the application after it is published to the server. When this check box is cleared, the initial state of the published application should be `stopped`. You can manually start the application by going into the Servers view, expand the server, right-click the application, and select **Start**.

If this check box is cleared, the application for running the Universal Test Client remains started on the server.

To configure the publishing settings for a WebSphere Application Server:

Procedure

1. In the Servers view, double-click your WebSphere Application Server to open the server editor.

2. Click the **Overview** tab.
3. Expand the **Publishing settings for WebSphere Application Server** section.
4. Use the radio buttons to select either:
 - Run server with resources within the workspace
 - Run server with resource on server
5. Optional: If you selected the **run server with resources within the workspace** radio button, you can select the **Minimize application files copied to the server** check box. When you are running a local WebSphere Application Server, this option is enabled by default.
6. Optional: If you want the workbench to make an automatic attempt to start the application after it is published to the server, select the **Automatically start applications after publishing** check box.
7. Save and close the editor.

- **Controlling incremental publish on a local WebSphere Application Server**

Use the **Advanced Publishing Settings** to control incremental publish on local installations of WebSphere Application Server based on updates to files with certain file extensions. You have the option of listing file extensions to either trigger or do not trigger the server to publish. You can define the settings specific to a server or as a workspace preference.

Parent topic: [Configuring a WebSphere Application Server](#)

Related concepts:

[Qualities of service for JAX-WS web services and clients](#)

Related tasks:

[Creating a WebSphere Application Server](#)

[Managing servers](#)

Related reference:

[Servers view](#)

[Publishing fails with duplicate class error](#)

Related information:

[Publishing your application](#)

Controlling incremental publish on a local WebSphere Application Server

Use the **Advanced Publishing Settings** to control incremental publish on local installations of WebSphere® Application Server based on updates to files with certain file extensions. You have the option of listing file extensions to either trigger or do not trigger the server to publish. You can define the settings specific to a server or as a workspace preference.

Before you begin

- To use the advanced publishing settings, you must enable the **Run server with resources within the workspace** publishing option. The advanced publishing settings are not available for the **Run server with resources on server** publishing option. For more information about these publishing options, see [Publishing settings for a WebSphere Application Server](#).

Important: Applicable edition: Full profile

About this task

Publishing a large application in its entirety can be slow, as a result incremental publish is a helpful feature that is provided by the development workbench. **Incremental publish** is when a server publishes updates to a previously published application quickly by publishing only those files that were modified. However, there are some files when modified that do not require the server to issue an incremental publish, which can take a long time for large applications. Therefore, you can improve publishing performance and shorten publishing cycles by specifying which file types need update during an incremental publish; or alternatively, specifying which file types do not require a publish.

Some file types that you might not be interested in publishing when modified include image or static web page files. You might want the server to avoid running an incremental publish when these file types are modified to minimize your development time. Here you can take advantage of the **List of file extensions that do not trigger the server to publish** option.

Alternatively, you might want to focus on developing certain parts of your application and only interested in seeing the updates on the server for these file types when modified. Here you can take advantage of the **List of file extensions that trigger the server to publish** option.

Procedure

1. In the Servers view, double-click your WebSphere Application Server to open the server editor.
2. On the **Overview** page under the **Publishing settings for WebSphere Application Server** section, select the **Run server with resources within the workspace** publishing option.
3. Click the **Set the Advanced Publishing Settings** link.
4. In the Advanced Publishing Settings window, select one of the following options:

Option	Description
--------	-------------

<p>Use the default settings from the Advanced Publishing Settings preference page</p>	<p>Select this option if you want to configure the server to use the same settings that are defined in the Advanced Publishing Settings preference page. You can find the Advanced Publishing Settings preference page by going to the toolbar and select Window > Preferences > Server > WebSphere Application Server > Advanced Publishing Settings. By default this option is selected. Select the Set Advanced Publishing Settings preference page link if you want to view or modify the Advanced Publishing Settings preference page. When you click this link, a dialog box opens automatically to the Advanced Publishing Settings preference page.</p>
<p>List of the file extensions that trigger the server to publish</p>	<p>Select this option if you want the server to trigger an incremental publish when there is an update to a file with one of the file extensions that are listed in the text box. In the text box, you can modify and type the list of file extensions by using a comma-separated list, such as in the following format: *.ext1, *.ext2, *.ext3 Any file extensions absent from the list cannot trigger a publish when the file is modified.</p>
<p>List of the file extensions that do not trigger the server to publish</p>	<p>Select this option if you do not want the server to trigger an incremental publish when there is an update to a file with one of the file extensions that are listed in the text box. In other words, the server remains unchanged and in its current state and status. In the text box, you can modify and type the list of file extensions by using a comma-separated list, such as in the following format: *.ext1, *.ext2, *.ext3 Any file extensions absent from the list triggers a publish when the file is modified.</p>

Tip: If essential file types, such as .class files, are excluded from triggering an incremental publish, then the server might not reflect the application changes as expected.

5. Optional: Click the **Restore Defaults** button if you want to reset to the default settings, including the list of file extensions in the text boxes. Here is the default list of file extensions that are defined in the text box for the **List of the file extensions that trigger the server to publish** option: *.class, *.xml, *.xmi, *.wsdl, *.jar, *.mf, *.xsd, *.composite, *.componentType, *.bp

Here is the default list of file extensions that are defined in the text box for the **List of the file extensions that do not trigger the server to publish** option: *.bmp, *.cab, *.css, *.doc, *.exe, *.gif, *.htm, *.html, *.ico, *.ini, *.jhtml, *.jpeg, *.jpg, *.js, *.jsp, *.jspf, *.jspx, *.jpg, *.pdf, *.png, *.properties, *.swf, *.tif, *.tiff, *.ttf, *.txt, *.xhtml, *.xls, *.zip

6. Click the **OK** button in the **Advanced Publishing Settings** window.
7. Type `Ctrl+s` to save the modifications in the server editor.

Parent topic: [Publishing settings for a WebSphere Application Server](#)

Specifying administrative settings to a secured WebSphere Application Server

If your WebSphere® Application Server runtime environment has global security that is enabled, you need to communicate the administrative settings from your development environment to the runtime server. On the workbench, you need to specify that security is enabled on the runtime environment, and provide the user name and password to the secured server. In addition, you need to establish a trust between the development workbench of this product and the server.

Before you begin



- The WebSphere Application Server runtime environment is secured. For information on configuring global security:
 - For WebSphere Application Server V8.5, see [Enabling security](#)
 - For WebSphere Application Server V8.0, see [Enabling security](#)
 - For WebSphere Application Server V7.0, see [Enabling security](#)

Important: Applicable edition: Full profile

About this task

To specify the administrative security settings to a secure local or remote WebSphere Application Server:

Procedure

1. In the Servers view, double-click your WebSphere Application Server. The server editor opens.
2. Click the **Overview** tab.
3. Expand the **Security** section.
4. Select the **Security is enabled on this server** check box. When this check box is selected, the server entry in the Servers view displays a lock icon. This icon is an example of a lock icon that displays when security is enabled for a WebSphere Application Server V7.0 (). However, if this check box is cleared, the remaining security settings are ignored, the server entry in the Servers view displays a warning icon, and the following warning message displays in the workbench: `Warning: The server is not secured. Consider enabling security on the server.`
This icon is an example of a warning icon that displays in the Servers view when security is disabled for WebSphere Application Server V7.0 (.
5. In the **User ID** field and **Password** fields, specify the user name and password for the current active administrative settings that are defined in the server configuration. **Windows** The specified user must have the **Log on as service** permission.
Linux The specified user must be logged on as root.
6. If you are working with a secured WebSphere Application Server, the **Automatically trust server certificate during SSL handshake** check box is by enabled by default. Each profile in the WebSphere Application Server environment contains a unique self-signed certificate that was created when the profile was created. . When a profile is federated to a deployment manager, the signer for that self-signed certificate is added to the common truststore for the cell. By default, clients (such as the development workbench) do not trust servers from different profiles in the WebSphere Application Server environment. That is, they do not contain the signer for these servers.
To help establishing this trust between the development workbench and the server, verify the **Automatically trust server certificate during SSL handshake** check box is selected. This check box specifies that when the workbench communicates to an administrative secured WebSphere Application Server, the server sends a signer certificate to the workbench. If the certificate is new, the workbench stores the certificate in its truststore file.
If the **Automatically trust server certificate during SSL handshake** check box is clear, the server status of the

Servers view displays the server as stopped and no connection can be made to the server. Make sure that you selected this check box, otherwise, you must manually establish the trust between the workbench and the administrative secured WebSphere Application Server, see **Manually exchanging signer certificates to establish a trust between the workbench and the server** topic for details.

7. Select **File > Save** to save the changes in the server editor.

Results

Note: When you are enabling administrative security for a server, do not give it a user ID that has the same name as the server where WebSphere Application Server is installed. Otherwise, the server might fail to start.

Parent topic: [Configuring a WebSphere Application Server](#)

Related tasks:

[Manually exchanging signer certificates to establish a trust between the workbench and the server](#)

Related reference:

[Problems working with a secured server using SSL connections](#)

Starting a remote WebSphere Application Server

Use the workbench to start a remote WebSphere® Application Server running on a Windows or Linux operating system. You can choose to use an operating system or Secure Shell (SSH) authentication method to access the remote host that is running the server. After you provide the workbench with authentication details to access the remote host, you can start the remote server in start or debug mode.

Before you begin

For Secure Shell (SSH) authentication method, the Linux operating systems generally support the use of SSH protocol. For Windows operating systems, however, you must install software from an independent software vendor to use SSH protocol, such as Cygwin. For details on how to generate and configure SSH key-based authentication, refer to your Secure Shell (SSH) documentation.

About this task


Important: Applicable edition: Full profile

Restriction:

- Starting a remote WebSphere Application Server by using the workbench is supported only on Linux and Windows operating systems. For all the other supported operating systems that are running WebSphere Application Server, you must start the server on the remote computer, and here are your options.
- Use the administrative console to set the correct VM parameters, for example, the debug parameter.
- Start the WebSphere Application Server on the remote computer. After the server is started on the computer, in the Servers view of the workbench, right-click the server, and select **Debug** or **Start** to restart the server in the mode that you want.

To enable the WebSphere Application Server to start remotely:

Procedure

1. In the Servers view, double-click the WebSphere Application Server you want to modify. The server editor opens.
2. Click the **Overview** tab.
3. In the **Host name** field under the **General Information** section, specify the fully qualified domain name server (DNS) name or IP address of the remote host machine where the server is running.
4. In the **Application server name** field, under the **Server** section, specify the logical name for the remote application server. For WebSphere Application Server, the logical name is unique and assigned to a server that distinguishes it from all other server instances within the node. This server name must already be created in the remote application server and its default setting is server1.
5. Under the **Remote Server Settings** section, select the **Enable the server to start remotely** check box.
6. Select whether your remote server is installed on a Windows or Linux operating system. **Note:** Remote server start uses RXA. For more information on RXA and RXA limitations for Windows and Linux, see **Requirements for using Remote Execution and Access (RXA)**.
7. In the **Server profile path** field, enter the location of the server profile. A profile is the set of files that define the runtime environment of a WebSphere Application Server. Here are examples of how to specify the path to the server profile, which depends on the operating system that is running the remote server; and for Windows operating system depends if you are going to select (in the next step) **Operating system authentication** or **Secure Shell (SSH)** as the logon method to access the remote host.  `/opt/<install_directory>/profiles/<profileName>` **Important:** You must have full access to the WebSphere Application Server (and the paths) on Linux systems, with read, write, and run authority. It is not sufficient to have authority to start and stop the server, or read and write access to specific files.

- **Operating system authentication:** <x>:\<install_directory>\profiles\<profileName>
- **Secure Shell (SSH):** /cygdrive/<x>/<install_directory>/profiles/<profileName> **Tip:** If you choose to use Cywin for your SSH protocol, you must prefix with /cygdrive/ followed by the drive letter; and use forward slashes (/) as the separator for the path.

- **<install_directory>**

- The directory where WebSphere Application Server is installed.

- **<profileName>**

- The name of the WebSphere Application Server profile. For example, the Installation Manager of this product creates a WebSphere Application Server v7.0 profile, which is assigned a default role with the following naming convention: was70profile1

- **<x>**

- The drive letter of the operating system where the profile of the WebSphere Application Server is located, such as C or D

8. Under the **Select a logon method to access the remote host** section, there are two options:
 - Select the **Operating system authentication** option, if you want to use a logon credential method of an operating system to access the remote server. If you select this option, enter the user name and password for the operating system that is running on the remote host.
 - Select the **Secure Shell (SSH)** option, if you want to use an SSH key-based authentication method to access the remote server. If you select this option, complete the following steps:
 - A. In the **Private key file** field, specify the location of the private key file. The default file name is `id_rsa` and a copy of this file must be available in the machine where the workbench is installed.
 - B. In the **User name** field, specify the user name to the remote computer, which contains an account that is configured with the SSH key-based authentication.
 - C. In the **Passphrase** field, specify the passphrase that is generated with the private key file. Typically, the passphrase is at least 16 characters long. If no passphrase is generated with the private key file, leave this field empty.
9. Type **Ctrl-S** to save the server configuration.
10. Close the editor.
11. In the Servers view, right-click the server. There are two options to start the server, depending on which mode you want to run the server:
 - **Start**
 - **Debug**

Parent topic: [Configuring a WebSphere Application Server](#)

Parent topic: [Starting a server](#)

Related tasks:

[Starting a server](#)

Advanced WebSphere Application Server configurations

This product serves as an administrative client to the WebSphere® Application Server, which focuses on assembling, publishing, and configuring server resources for Java™ EE applications target to a WebSphere Application Server. However, when you are working with this product you might find that you need to use advanced server configurations to change the behavior of the WebSphere Application Server. These advanced server configurations are often configured by using another administrative client, such as the WebSphere Application Server administrative console.

Important: Applicable edition: Full profile

Some common tasks are provided. They are related to advanced WebSphere Application Server configurations that you might encounter while you are working with this product. When possible, links are provided to the WebSphere Application Server information Center (<http://www.ibm.com/software/webservers/appserv/was/library/>) where you find the complete documentation on advanced server configuration for WebSphere Application Server.

Examples of advanced WebSphere Application Server configurations

- [Configuring a WebSphere Application Server with Federal Information Processing Standard enabled](#)
- [Configuring Lightweight Directory Access Protocol user registries](#)
- [Configuring port settings](#)

Parent topic: [Configuring a WebSphere Application Server](#)

Configuring a WebSphere Application Server with Federal Information Processing Standard enabled

You can set up the workbench to connect to a WebSphere® Application Server with Federal information Processing Standard (FIPS) enabled.

Before you begin

- Configure a WebSphere Application Server with FIPS, details are available in the information Center for the WebSphere Application Server.
- For WebSphere Application Server V8.5, see [Configuring Federal Information Processing Standard Java™ Secure Socket Extension files](#) topic.
- For WebSphere Application Server V8.0, see [Configuring Federal Information Processing Standard Java Secure Socket Extension files](#) topic.
- For WebSphere Application Server V7.0, see [Configuring Federal Information Processing Standard Java Secure Socket Extension files](#) topic.

Important: Applicable edition: Full profile

About this task

To connect the workbench to a WebSphere Application Server with FIPS enabled:

Procedure

1. Verify that you have the correct server configuration to enable FIPS:
 - A. In addition to enabling the FIPS option on the server, such as selecting the **Use the Federal Information Processing Standard (FIPS)** option in the administrative console of WebSphere Application Server, verify that you configured the `sas.client.props` and `soap.client.props` files correctly on the server. **Important:** The configuration changes to the `sas.client.props` and `soap.client.props` files must be done at the profile-level and not at the server-level. Modify the `sas.client.props` and `soap.client.props` files that are found in the profile-level at `y:/profiles/profile_name/properties` directory. Do not modify the `soap.client.props` file that is found in the server-level at `y:/properties` directory, where `y` is the installation directory of WebSphere Application Server. For configuration details, see the [Configuring Federal Information Processing Standard Java Secure Socket Extension files](#) topic links available in the **Before you begin** section. **Tip:** If the message `FIPS is enabled` displays in the `SystemOut.log` file, which is in the `y:/profiles/profile_name/logs/server_name` directory, where `y` is the installation directory of WebSphere Application Server, you successfully enabled FIPS on the server.
2. If you are connecting to a remote server, verify that you made the same changes to the `sas.client.props`, `soap.client.props`, and `java.security` files in the following two locations.
 - The server files, as instructed in the **Before you begin** section and the previous steps.
 - The workbench files: The `sas.client.props` and `soap.client.props` files are in the `x:/runtimes/base_v<y>_>_stub/profiles/profile_name/properties` directory. The `java.security` file is in the `x:/runtimes/base_v<y>_>_stub/java/jre/lib/security` directory, where `x` is the installation directory of this product and `<y>` is the version-level of the WebSphere Application Server.
3. Open the workbench of the product.
4. [Start the WebSphere Application Server.](#)

Parent topic: [Configuring a WebSphere Application Server](#)

Managing servers

You can use the server tools views to manage servers. You can start and stop servers, deactivate and activate servers, and manage a server configuration.

- **Starting a server**

When you are ready to test your projects, you must start a server. In some situations, the server tools start a server for you automatically.

- **Stopping a server**

You can stop the server from the Servers view.

- **Deactivating and activating your server**

You can deactivate and activate your server by using the menu in the servers view.

- **Accessing the administrative console**

You can use the advanced functions that are provided in the administrative console to change settings in your server configuration.

- **Running administrative script files on WebSphere Application Server**

You can run administrative scripts from within the development environment, without having to switch to the non-graphical command interpreter, WebSphere Application Server wsadmin tool. Use the WebSphere® Application Server Administrative Script Launcher to run administrative script files on WebSphere Application Server. To run script files within the development environment you need to specify the location of your script, the runtime environment to interpret your script, and the security settings; if your script is running against a secured server.

- **Keeping the WebSphere Application Server running after exiting the development environment**

To specify that the server will continue running after you shut down the development environment.

Parent topic: [Testing and publishing on a server](#)

Related tasks:

[Publishing settings for a WebSphere Application Server](#)

Related information:


- [🔗 Setting timeout on the start or stop of the server](#)
- [🔗 Setting a default server](#)
- [🔗 Clean projects published on the server](#)
- [🔗 Displaying or hiding the metadata of the server](#)

Starting a server

When you are ready to test your projects, you must start a server. In some situations, the server tools start a server for you automatically.

Before you begin

- [Create a server](#)

 If you are trying to run the WebSphere® Application Server other than the integrated test environment as a non-root user ID, this might fail. By default, WebSphere Application Server uses the root user ID to run the server. A non-root user profile must be created by the root user. Depending on the assignment of profile directory ownership that is completed by the installer, a non-root user can create a profile, start WebSphere Application Server, or do both.

- For WebSphere Application Server V8.5, see the [Managing profiles for non-root users](#) topic
 - For WebSphere Application Server V8.0, see the [Managing profiles for non-root users](#) topic
 - For WebSphere Application Server V7.0, see the [Managing profiles for non-root users](#) topic
- Starting a remote WebSphere Application Server that uses the workbench is supported only on Linux and Windows operating systems. For all the other supported operating systems that run WebSphere Application Server, you must manually start the server on the remote computer. For more information, see [Starting a remote WebSphere Application Server](#).

Important: Applicable edition: Full profile

About this task

Once you start the server, you can test the projects that are associated with the server. You can stop and restart the server at any time. The status of the server is displayed in the **Status** column of the Servers view.

Procedure

1. Switch to the Servers view.
2. In the Servers view, right-click the server that you want to start.
3. Select **Start**. The following things happen:
 - If you selected the **Automatically publish when starting servers** check box on the Server preferences page (Window > Preferences > Server), the server tools check to see if your project and files on the server are synchronized before you start the server. If they are not, the project and the files are automatically updated on the remote server when it is started.
 - A Console view opens in the workbench. It takes a minute to start the server. If the server fails to start, check the reason that it failed in the Console.
 - In the **Status** column of the Servers view, the status of the server changes to *Started*.

Results

Tip: The server is automatically started when you right-click on a file and then select **Run As > Run on Server**.

Parent topic: [Managing servers](#)

Related tasks:

[Starting a remote WebSphere Application Server](#)

[Optimizing starting the WebSphere Application Server for development](#)

Related reference:

A timeout error displays when starting the server immediately after a stop request is issued

Related information:

[Starting a server in debug mode](#)

Stopping a server

You can stop the server from the Servers view.

Before you begin

- For WebSphere® Application Server, wait for the server to start completely before you stop the server. **Restriction:** When you try to stop a WebSphere Application Server while the server is starting, the server continues to start and ignore the stop request. The status of the server in the Servers view, might change from **Starting** to **Stopped**. But when the server finishes starting, the status of the server changes back to **Started**. You must wait for the server to start completely before you stop the server.
- For remote WebSphere Application Server, you can stop the remote server from this workbench. However, starting a remote WebSphere Application Server by using the workbench is supported only on Linux and Windows operating systems. For all the other supported operating systems that run WebSphere Application Server, you must manually start the server on the remote computer when the server is stopped. For more details on how to start the WebSphere Application Server remotely on Linux or Windows, see the [Starting a remote WebSphere Application Server](#) topic. If the remote WebSphere Application Server needs restarting, you can right-click the server from the Servers view and select **Restart**.

If the remote WebSphere Application Server is shared by multiple users, then stopping the remote server stops the shared server, which makes it unavailable for other users to use.

Important: Applicable edition: Full profile

Procedure

1. In the **Servers** view (> **Window** > **Show View** > **Other** > **Server** > **Servers**), click **OK** to select the server that you want to stop.
2. Click the **Stop the server** icon in the toolbar. In the **Servers** view, the status of the server changes to **Stopped**.
3. If the server fails to stop, you can terminate the process by:
 - A. Switching to the **Debug** perspective
 - B. In the **Process** view, select the server process that you want to stop.
 - C. Click the **Terminate** icon in the toolbar.

Note: When terminating a server, the server process will end and the server will not go through the normal routine of stopping, for example calling the destroy() method on a servlet.

Parent topic: [Managing servers](#)

Related reference:

[A timeout error displays when starting the server immediately after a stop request is issued](#)

Deactivating and activating your server

You can deactivate and activate your server by using the menu in the servers view.

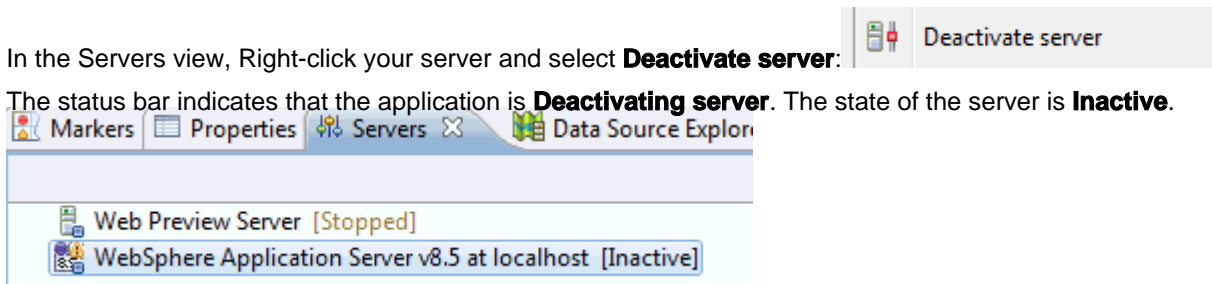
About this task

Important: Applicable edition: Full profile

You can deactivate servers that you do not currently use on the Servers view. Deactivating servers reduces memory consumption and network traffic that is used by the tool. To deactivate your server:

Procedure

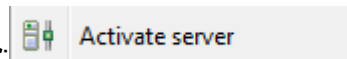
1. In the Servers view, Right-click your server and select **Deactivate server**:
2. The status bar indicates that the application is **Deactivating server**. The state of the server is **Inactive**.



What to do next

To activate an inactive server:

1. In the Servers view, Right-click your server and select **Activate server**:



Parent topic: [Managing servers](#)

Accessing the administrative console

You can use the advanced functions that are provided in the administrative console to change settings in your server configuration.

About this task

The WebSphere® Application Server no longer needs you to enable the administrative console in the server editor. The development environment no longer has the server configuration in the workspace. By default, the administrative console is enabled in the default profile of the server.

Important: Applicable edition: Full profile

To access the administrative console:

Procedure

1. [Start the WebSphere Application Server](#).
2. In the Servers view, right-click the server, and then select **Administration > Run administrative console**. The administrative console logon window opens in the Web Browser view.
3. If the server is secured, specify the user ID and password to access the administrative console.
4. Click the **Log in** button.

Results

After changes are made, you must restart the server.

Related information:

[WebSphere Application Server Information Center: Using the administrative console](#)

Running administrative script files on WebSphere Application Server

You can run administrative scripts from within the development environment, without having to switch to the non-graphical command interpreter, WebSphere Application Server wsadmin tool. Use the WebSphere® Application Server Administrative Script Launcher to run administrative script files on WebSphere Application Server. To run script files within the development environment you need to specify the location of your script, the runtime environment to interpret your script, and the security settings; if your script is running against a secured server.

Before you begin

- Define an administrative script file with the appropriate file extensions. A Jython script file requires the `.py`, `.PY`, `.jy`, or `.JY` file extension. A Jacl script file requires the file extension `.jacl`.
- For running administrative scripts, the workbench requires a local installation of WebSphere Application Server, even if you are running the administrative scripts against a remote server. The reason is because the workbench uses the administrative scripting run times that are provided in the libraries of the locally installed server. **Restriction:** If the local WebSphere Application Server was installed by a root user and you attempt to run the WebSphere Application Server Administrative Script Launcher while you run the workbench with a non-root user ID, you are going to encounter a similar error message in the console view:

```
!MESSAGE Error reading configuration: /opt/IBM/WAS_install_directory
/profiles/unsecure/configuration/org.eclipse.osgi/.manager/.fileTableLock (Permission denied)
!STACK 0
java.io.FileNotFoundException: /opt/IBM/WAS_install_directory
/profiles/unsecure/configuration/org.eclipse.osgi/.manager/.fileTableLock (Permission denied)
```


The error occurs because the WebSphere Application Server Administrative Script Launcher attempts to write data into the WebSphere Application Server profile, which can be written to only by the root user. The non-root user does not have the correct file permissions for writing into a root profile. The workaround is to run the WebSphere Application Server Administrative Script Launcher with a non-root profile. You can select a non-root profile by using the **Profile name** list available in the WebSphere Application Server Administrative Script Launcher. For more information on creating a non-root profile, see the [Creating profiles for non-root users](#) topic available in the WebSphere Application Server documentation.

- Start the local or remote server you want to run your administrative script file against.

About this task

Run administrative script files on WebSphere Application Server.

Procedure

1. You can open the WebSphere Application Server Administrative Script Launcher in any of the following ways:
 - In the Servers view, right-click a WebSphere Application Server and select **Administration > Run administrative script**
 - Right-click on any text editor, for example the Jython editor, and select **Run As > Administrative Script**
 - In the Enterprise Explorer or the Navigator view, right-click a file with a valid file extension, and select **Run As > Administrative Script**
 - In the toolbar of the Java™ or Java EE perspective, select **Run > Run Configurations > WebSphere Application Server Administrative Script** and then select **New launch configuration** ()

2. The **Script** page, is the main page of the WebSphere Application Server Administrative Script Launcher.
 3. In the **Administrative Script** field, complete either the following to specify the location of your script file:
 - Type the path to your script file.
 - Click **Workspace** to browse to your script file on your workspace.
 - Click the **File System** to browse to your script file on your file system.
 - Click **Variables** to automatically expand out the path to your script file when a resource is selected on your workspace.
 4. In the **Scripting runtime environment** list, select a name of an installed WebSphere Application Server runtime environment as defined in the Preferences page (**Window > Preferences > Server > Runtime Environments**). Specify a runtime environment that is on your local file system to interpret your script file. If the runtime environment you want to interpret your script file is not available in the drop-down list, click **New** to define a new WebSphere Application Server runtime environment. The New Server Runtime Environments wizard opens. Although you can create any runtime environments with the New Server Runtime Environments wizard, the WebSphere Application Server Administrative Script Launcher supports only the following local runtime environments:
 - WebSphere Application Server v6.1 or later
 As a result, the **Scripting runtime environment** drop-down list is going to display only the supported runtime environments.
 5. In the **Profile name** list, select a profile that belongs to the previously selected runtime environment. A profile is the set of files that define the runtime environment. If you select the **[Unspecified]** option, the tool is going to choose the profile who is assigned the default role.
 6. In the **wsadmin arguments** field, specify advanced command-line invocation syntax for the wsadmin scripting client. A command-line invocation syntax reference is available in the WebSphere Application documentation, see [Wsadmin Tool](#) topic for more details. This WebSphere Application Server Administrative Script Launcher already handles the following information that is passed to the wsadmin scripting client:
 - The file name and path of the script file
 - The language of the script file
 - The name of the WebSphere Application Server profile that is used to run your script file
 - The user ID and password to authenticate to the targeted secured server
 Do not specify the information again in the **wsadmin arguments** field. If you are running the administrative scripts against a remote server, specify whether you are using either a remote method invocation (RMI) or SOAP connection type and its port value. If security is enabled on the remote server, specify the **-user** and **-password** parameters. Use one of the following command syntaxes in the **wsadmin arguments** field:
 - For SOAP connection type: `-conntype SOAP [-host host_name] [-port port_number] [-user userid] [-password password]`
 For example: `-conntype SOAP -host mycomputer.mycompany.com -port 8800 -user myUserID -password myPassword`
 - For RMI connection type: `-conntype RMI [-host host_name] [-port port_number] [-user userid] [-password password]`
 For example: `-conntype RMI -host mycomputer.mycompany.com -port 2809 -user myUserID -password myPassword`
- Tip:** If you want to pass arguments to your Jython script, do not specify them in this wsadmin arguments field, as this field is target for the wsadmin scripting client. Instead, select the **Arguments** page and in the **Program arguments** text box specify your program arguments.
- Note:** If you need to specify the program arguments for running your Jython scripts, keep in mind that the first program parameter is used for different purposes when debugging versus running the Jython scripts.
7. In the **Security** section, specify whether your script is running against a server that is secured by selecting one of the following options.

Option	Description
--------	-------------

<p>No security on target server</p>	<p>Specifies your script is running against a server that is not secured.</p>
<p>As defined in soap.client.props or sas.client.props file</p>	<p>Specifies that your script is running against a secured WebSphere Application Server, and you supplied authentication information in the <code>sas.client.props</code> or the <code>soap.client.props</code> files to communicate with this secured server.</p> <p>The <code>sas.client.props</code> and the <code>soap.client.props</code> files are in the properties directory for each WebSphere Application Server profile, for example <code>x:/profilePath/properties</code>.</p> <p>If you use a Remote Method Invocation (RMI) connector, set values for the following properties in the <code>sas.client.props</code> file with the appropriate values.</p> <pre>com.ibm.CORBA.loginUserId= com.ibm.CORBA.loginPassword=</pre> <p>Also, set the following property:</p> <pre>com.ibm.CORBA.loginSource=properties</pre> <p>The default value for this property is <code>prompt</code> in the <code>sas.client.props</code> file. If you leave the default value, a dialog box appears with a password prompt. If the script is running unattended, it appears to hang. If you use a Simple Object Access Protocol (SOAP) connector, set values for the following properties in the <code>soap.client.props</code> file with the appropriate values: <pre>com.ibm.SOAP.securityEnabled=true com.ibm.SOAP.loginUserId= com.ibm.SOAP.loginPassword=</pre> <p>Optionally, set the following property: <code>com.ibm.SOAP.loginSource=none</code></p> <p>You can find the default value for this property in the <code>soap.client.props</code> file. If you accept the default value and do not provide <code>loginUserId</code> and <code>loginPassword</code> values, a dialog box appears with a password prompt. If the script is running unattended, it appears to hang. Note: If you specify user and password in the Specify section and in the <code>sas.client.props</code> file or the <code>soap.client.props</code> file, the WebSphere Application Server Administrative Script Launcher overrides the information in the <code>.props</code> file.</p> </p>

<p>Specify</p>	<p>Specifies that your script is running against a server that is secured. In the User ID and Password fields, type the user ID and password to authenticate to this targeted secured server.</p> <p>If you specify user ID and password information in this section and in the <code>soap.client.props</code> file or the <code>soap.client.props</code> file, the WebSphere Application Server Administrative Script Launcher overrides the information in the <code>.props</code> file.</p> <p>Attention: The use of the password field can result in security exposure as the password information becomes visible to the system status program such as <code>ps</code> command, which can be started by other users to display all the running processes. Do not use this option if security exposure is a concern. Instead, select As defined in <code>soap.client.props</code> or <code>soap.client.props</code> file; and specify the user ID and password information in the <code>soap.client.props</code> file for SOAP connector or <code>soap.client.props</code> file for RMI connector. The <code>soap.client.props</code> and <code>soap.client.props</code> files are in the properties directory of your WebSphere Application Server profile.</p>
-----------------------	---

8. Click **Apply** to save your configuration. The appropriate VM arguments and class paths for your selected WebSphere Application Server runtime environment are automatically populated. **Tip:** If you change your selection for the **Scripting runtime environment** list, click **Apply** for each change to automatically populate the correct VM arguments and class paths for your newly selected WebSphere Application Server runtime environment.
9. Click **Run** to run your script file.

- [Debugging Jython script files on WebSphere Application Server](#)

You can detect and diagnose errors in your Jython script with the debugger. You can control the running of your script by setting breakpoints, suspending threads, stepping through the code, and examining the contents of the variables. To debug Jython scripts, use the WebSphere Application Server Administrative Script Launcher to set up the runtime environment to interpret your script, and the security settings; if your script is running against a secured server.

Parent topic: [Managing servers](#)

Debugging Jython script files on WebSphere Application Server

You can detect and diagnose errors in your Jython script with the debugger. You can control the running of your script by setting breakpoints, suspending threads, stepping through the code, and examining the contents of the variables. To debug Jython scripts, use the WebSphere® Application Server Administrative Script Launcher to set up the runtime environment to interpret your script, and the security settings; if your script is running against a secured server.

Before you begin

- For debugging administrative scripts, the workbench requires a local installation of WebSphere Application Server v6.1 or later even if you are debugging the administrative scripts against a remote server. The reason is that the workbench uses the administrative scripting runtime environments that are provided in the libraries of a locally installed server.
- Start the local or remote server you want to debug your administrative script file against.

Restriction: WebSphere Application Server v6.1 and later is the only supported runtime environment for debugging Jython script files. However, when you run (as opposed to debugging) Jython script files, there is a different list of supported runtime environments. See [Running administrative script files on WebSphere Application Server](#) for details.

Tip: The difference between running versus debugging Jython scripts is that for debugging Jython scripts you can set breakpoints in the script; whereas in running a Jython script you are running the script without interruptions.

Restriction: To debug Jython scripts, the Jython scripts must exist within the workbench by either creating or importing the Jython script into the workbench. You must debug your Jython script within the workbench, as opposed to running a Jython script located on the file system. Debugging tools such as the breakpoint margin to set breakpoints is not available when you debug the Jython script outside the workbench on the file system.

About this task

To debug Jython scripts:

Procedure

1. Open a Jython script file in the Jython editor:
 - A. In Navigator view, expand the Jython project that contains the Jython script file you want to debug your Jython script file.
 - B. Right-click the Jython script file.
 - C. In the menu, select **Open With > Jython Editor**.
2. Set breakpoints in the Jython script. To set one or more breakpoints in the Jython editor, double-click in the margin next to the line of code that you want to set as a breakpoint.
3. In the Jython editor, right-click and in the menu select **Debug As > Administrative script**. The WebSphere Application Server Administrative Script Launcher opens. Use the WebSphere Application Server Administrative Script Launcher to debug administrative script files on WebSphere Application Server from within the development environment.
4. The **Script** page, is the main page of the WebSphere Application Server Administrative Script Launcher.
5. The **Administrative Script** field is automatically pre-filled with the path to your script file.
6. In the **Scripting runtime environment** list, select a name of a WebSphere Application Server runtime environment as defined in the Preferences page (**Window > Preferences > Server > Runtime Environments**). You must specify a runtime environment that is on your local file system to interpret your script file. If the runtime environment that you want to interpret your script file is not available in the list, click **New** to define a new WebSphere Application Server runtime environment. The New Server Runtime Environment wizard opens. You can create any runtime environments with the New Server Runtime Environment wizard. However, when you use the WebSphere Application Server

Administrative Script Launcher for debugging Jython script, only the local WebSphere Application Server v6.1 or later runtime environment is supported for debugging.

7. In the **Profile name** list, select a profile that belongs to the previously selected runtime environment. A profile is the set of files that define the runtime environment. If you select the **[Unspecified]** option, the tool is going to choose the profile who is assigned the default role.
8. In the **wsadmin arguments** field, specify advanced command-line invocation syntax for the wsadmin scripting client. A command-line invocation syntax reference is available in the WebSphere Application documentation, see [Wsadmin Tool](#) topic for more details. This WebSphere Application Server Administrative Script Launcher already handles the following information that is passed to the wsadmin scripting client:

- The file name and path of the script file
- The language of the script file
- The name of the WebSphere Application Server profile that is used to run your script file
- The user ID and password to authenticate to the targeted secured server

Do not specify the information again in the **wsadmin arguments** field. If you are running the administrative scripts against a remote server, specify whether you are using either a remote method invocation (RMI) or SOAP connection type and its port value. If security is enabled on the remote server, specify the **-user** and **-password** parameters. Use one of the following command syntaxes in the **wsadmin arguments** field:

- For SOAP connection type: `-conntype SOAP [-host host_name] [-port port_number] [-user userid] [-password password]`

For example: `-conntype SOAP -host mycomputer.mycompany.com -port 8800 -user myUserID -password myPassword`

- For RMI connection type: `-conntype RMI [-host host_name] [-port port_number] [-user userid] [-password password]`

For example: `-conntype RMI -host mycomputer.mycompany.com -port 2809 -user myUserID -password myPassword`

Tip: If you want to pass arguments to your Jython script, do not specify them in this wsadmin arguments field, as this field is target for the wsadmin scripting client. Instead, select the **Arguments** page and in the **Program arguments** text box specify your program arguments.

Note: If you need to specify the program arguments for running your Jython scripts, keep in mind that the first program parameter is used for different purposes when debugging versus running the Jython scripts.

9. In the **Security** section, specify whether your script is running against a server that is secured by selecting one of the following options.

Option	Description
No security on target server	Specifies your script is running against a server that is not secured.

<p>As defined in soap.client.props or sas.client.props file</p>	<p>Specifies that your script is running against a secured WebSphere Application Server, and you supplied authentication information in the <code>sas.client.props</code> or the <code>soap.client.props</code> files to communicate with this secured server.</p> <p>The <code>sas.client.props</code> and the <code>soap.client.props</code> files are in the <code>properties</code> directory for each WebSphere Application Server profile, for example <code>x:/profilePath/properties</code>.</p> <p>If you use a Remote Method Invocation (RMI) connector, set values for the following properties in the <code>sas.client.props</code> file with the appropriate values.</p> <pre>com.ibm.CORBA.loginUserId= com.ibm.CORBA.loginPassword=</pre> <p>Also, set the following property:</p> <pre>com.ibm.CORBA.loginSource=properties</pre> <p>The default value for this property is <code>prompt</code> in the <code>sas.client.props</code> file. If you leave the default value, a dialog box appears with a password prompt. If the script is running unattended, it appears to hang. If you use a Simple Object Access Protocol (SOAP) connector, set values for the following properties in the <code>soap.client.props</code> file with the appropriate values: <pre>com.ibm.SOAP.securityEnabled=true com.ibm.SOAP.loginUserId= com.ibm.SOAP.loginPassword=</pre> <p>Optionally, set the following property:</p> <pre>com.ibm.SOAP.loginSource=none</pre> <p>You can find the default value for this property in the <code>soap.client.props</code> file. If you accept the default value and do not provide <code>loginUserId</code> and <code>loginPassword</code> values, a dialog box appears with a password prompt. If the script is running unattended, it appears to hang. Note: If you specify user and password in the <code>Specify</code> section and in the <code>sas.client.props</code> file or the <code>soap.client.props</code> file, the WebSphere Application Server Administrative Script Launcher overrides the information in the <code>.props</code> file.</p> </p>
--	---

<p>Specify</p>	<p>Specifies that your script is running against a server that is secured. In the User ID and Password fields, type the user ID and password to authenticate to this targeted secured server.</p> <p>If you specify user ID and password information in this section and in the <code>sas.client.props</code> file or the <code>soap.client.props</code> file, the WebSphere Application Server Administrative Script Launcher overrides the information in the <code>.props</code> file.</p> <p>Attention: The use of the password field can result in security exposure as the password information becomes visible to the system status program such as <code>ps</code> command, which can be started by other users to display all the running processes. Do not use this option if security exposure is a concern. Instead, select As defined in <code>soap.client.props</code> or <code>sas.client.props</code> file; and specify the user ID and password information in the <code>soap.client.props</code> file for SOAP connector or <code>sas.client.props</code> file for RMI connector. The <code>soap.client.props</code> and <code>sas.client.props</code> files are in the properties directory of your WebSphere Application Server profile.</p>
-----------------------	---

10. Click **Apply** to save your configuration. The appropriate VM arguments and class paths for your selected WebSphere Application Server runtime environment are automatically populated. **Tip:** If you change your selection for the **Scripting runtime environment** list, click **Apply** for each change to automatically populate the correct VM arguments and class paths for your newly selected WebSphere Application Server runtime environment.
11. Verify that you have breakpoints set in the Jython editor (specified in the earlier steps). Otherwise, your debug session runs to completion.
12. Click **Debug** to debug your script. The Debug perspective opens.

What to do next

The debugger suspends on the set breakpoint on your Jython script, and you can further debug the script in the Debug perspective.

Parent topic: [Running administrative script files on WebSphere Application Server](#)

Testing applications on a server

You can use the workbench to test one or more projects on a server.

- [Testing artifacts on a server](#)

You can use the workbench to test one or more application artifacts on a server.

- [Testing on a Java Platform, Enterprise Edition Preview server](#)

Use the Java™ Platform, Enterprise Edition Preview server when you want to quickly compile, test, and run resources in a Java EE web project. The Java EE web projects that you can test on a Java Platform, Enterprise Edition Preview server are static web projects, dynamic web projects, and utility projects. The term Java EE web project refers to any of these projects.

Parent topic: [Testing and publishing on a server](#)

Related information:

[↗ Testing on an HTTP Preview server](#)

Testing artifacts on a server

You can use the workbench to test one of more application artifacts on a server.

About this task

To test artifacts on a server:

Procedure

1. In the Enterprise Explorer view from the Java™ EE perspective, expand the project that contains the file you want to test.
2. Right-click the artifact that you want to test. Examples of artifacts include the Web project or a file within the Web project (such as a JSP file, servlet, or HTML file), or an enterprise bean.
3. Select **Run As > Run on Server**. The Server selection wizard appears.
4. Use the radio buttons to specify how you want to select the server. Select from the following options:
 - **Choose an existing server**: This option uses an existing server that is defined in your workbench.
 - **Manually define a server**: This option creates a new server.
5. Under the **Select the server that you want to use** list, select a server that you want to run your application. The list of available servers, under the **Select the server that you want to use** list, depends on the Java EE specification level you defined for the artifact and the supported servers on this workbench.
6. Optional: Click **Next** to configure extra settings, such as configuring settings specific to the server, and adding or removing projects that are configured on the server.
7. Click **Finish**. The server tools automatically complete the following tasks for you:
 - Creates the server and adds it to the Servers View.
 - Adds your project to the server.
 - Starts the server. (Depending on the server settings, starting the server might take time.)
 - If the **Automatically publish when starting servers** check box on the Server preferences page (**Window > Preferences > Server > Launching**) is selected, the workbench checks to see if your project and files on the server are synchronized. If they are not, the project and the files are automatically updated when the server is restarted.
 - Depending on the artifact you are testing, the output of running your artifact might display in the Console view or in a Web browser that automatically opens.

Parent topic: [Testing applications on a server](#)

Debugging applications on a server

You can debug applications on a server to detect and diagnose errors in your application. Use the TCP/IP monitor view to view data sent and received over the ports of the server. In addition, use the debugging tools in the workbench to control running your application on the server by setting breakpoints, suspending threads, stepping through the code, and examining the contents of the variables. You can debug artifacts, such as JSP files, on a server without losing the state of your application.

- **Hot method replace when debugging applications on WebSphere Application Server**

If you want to change to your Java™ classes while you are debugging, the hot method replace is automatically enabled when you are running an application in debug mode on WebSphere® Application Server. Hot method replace allows most application changes to pick up automatically without requiring an application or server restart.

- **Debugging a JSP file on a server**

You can use the debugger to detect and diagnose errors in your application. You can control the execution of your program by setting breakpoints, suspending threads, stepping through the code, and examining the contents of the variables. You can debug a JavaServer Page (JSP) without losing the state of your application.

Parent topic: [Testing and publishing on a server](#)

Related information:

[Debugging a servlet on a server](#)

[Monitoring server ports](#)

Hot method replace when debugging applications on WebSphere Application Server

If you want to change to your Java™ classes while you are debugging, the hot method replace is automatically enabled when you are running an application in debug mode on WebSphere® Application Server. Hot method replace allows most application changes to pick up automatically without requiring an application or server restart.

Tips about hot method replace:

- [Full profile] Hot-method replace cannot be used while you are profiling an application. When you profile an application, data that is related to the runtime behavior of a program is collected and presented in both graphical and tabular views. Using profiling, you can see which operations take the most time and helps you to find and solve memory leaks. For more information, see the Profiling documentation.
- [Full profile] When you are using hot method replace to debug servlets, the server reloads a method that is changed. It does not reload the class or reload and reinitialize the servlet.
- [Full profile] JSP debugging does not support hot method replace.
- [Full profile] When hot method replace automatically running, the JIT compiler is also enabled. The JIT compiler that is used when hot method replace is running, is much faster than when you are using normal debugging. The `JAVA_COMPILER=NONE` environment variable and system property `-Djava.compiler=NONE` to disable the JIT compiler is going to be ignored if hot method replace is running when debugging an application. When hot method replace is running, the `-Xint` option is going to disable the JIT compiler.
- [Full profile] If **Run -Xquickstart in the Java virtual machine settings** check box is selected in the server editor, the workbench ignores this option when the server starts in debug mode. The `-Xquickstart` JVM setting on startup does not run when the server starts in debug mode. For more information, see *Optimize starting the WebSphere Application Server for development*
- **8.5.5.6** When a Liberty server is running in debug mode, any modifications to method annotations are not automatically updated on the server. In the following example, modifications to the `@onMessage` method annotation are not updated automatically, because hot method replace is enabled by default in debug mode. Changes to method annotations are not handled by hot method replace. Publishing is required to load the changes.`@onMessage`

```
public void receiveMessage(String message) {  
    try {  
        count++;  
        .....  
    }  
}
```

Parent topic: [Debugging applications on a server](#)

Related concepts:

[When the test server requires restarting](#)

[When to wait for the time interval of an automatic publishing to pass on a WebSphere Application Server](#)

Related tasks:

[Optimizing starting the WebSphere Application Server for development](#)

Debugging a JSP file on a server

You can use the debugger to detect and diagnose errors in your application. You can control the execution of your program by setting breakpoints, suspending threads, stepping through the code, and examining the contents of the variables. You can debug a JavaServer Page (JSP) without losing the state of your application.

Procedure

See [Debugging a JSP file on a server](#). In addition, you can set one or more breakpoints by completing the following steps:

1. Verify that you are using the Source page of the editor. If multiple pages are available for the editor, such as Design, Source, or Preview page, select the **Source** page. If multiple pages are not available, the editor defaults to a source editor.
2. Select a line of code in the editor and double-click where you want to add the breakpoint in the area of the marker bar.

Results

Note: If you are running the JSP file on a remote WebSphere® Application Server, the JSP cache directory must be cleared to force the class file to be in the correct JSP source debug mode. If the cache is not cleared, some compiled unchanged JSP files might still be compiled in the previous JSP source debug mode.

Parent topic: [Debugging applications on a server](#)

When to wait for the time interval of an automatic publishing to pass on a WebSphere Application Server

You can set an interval of time of when changes to the files that run on the server are automatically reloaded to the server. The following subtopics describe different situations when you might be waiting for the interval of time to pass for the workbench to issue an automatic publish command on a WebSphere® Application Server. In some cases, you do not need to wait for the interval of time to pass when the changes to the applications are dynamically reloaded to the server, which depends on the type of resource that is modified, the location of the server and its publishing setting option, and the mode you are running the server. The table summarizes these situations.

Automatically publishing to a server

Important: Applicable edition: Full profile

If the **Automatically publish when starting servers** check box on the Server preferences page (**Window > Preferences > Server > Launching**) is selected, the workbench checks to see if your project and files on the server are synchronized. If they are not, the project and the files are automatically updated when the server is either started or restarted.

In the workbench, you have several options to choose for the **Publishing** settings. You can set these Publishing settings by going into the Servers view, right-click the server and select **Open**. The Server editor opens. In the Overview page of the server editor, under the **Publishing** settings, you are going to find the following settings:

- **Never publish automatically:** Specifies that the workbench never publishes files to the server.
- **Automatically publish when resources change:** Specifies the workbench to issue a publish after changes on a file that is associated to the server are saved and a full-time interval that is passed in the **Publishing interval** setting.
- **Automatically publish after a build event:** Specifies the workbench to issue a publish after changes on a file that requires a build and is associated to the server are saved, and a full-time interval that is passed in the **Publishing interval** setting.
- **Publishing interval (in seconds):** Specifies the number of seconds that needs to pass before the workbench calls a publish to happen on the server. However, if you make a subsequent change to the files before this time interval has completed, the publish is delayed as the timer is reset. The workbench makes a publish to the server only after the full-time interval passed. If you set the publishing interval to 0 seconds, an immediate publish happens after changes on a file are saved.

In the workbench, the default setting is the **Automatically publish when resources change** option is enabled with a value set in the publishing interval.

Manually publishing to a server

If you do not want to wait for the automatic publishing interval to pass, at anytime you can manually request the workbench to issue a publish command to the server. Each manual publish command causes a single publishing request to the server. To publish your application manually, you can complete one of the following in the Servers view:

- Select the server and then click the **Publish to the server** icon that is on the toolbar.
- Right-click the server and then select **Publish**.

Location of the server and its publishing settings

The location of the server whether it is a remote or local server, together with its publishing settings whether it is using **Run server with resources on Server** or **Run server with resources within the workspace** option, are factors that define if you require to wait for the interval of time to pass for the workbench to issue an automatic publish command on a WebSphere Application Server. Keep in mind, you can at anytime issue a manual publish request to eliminate the wait, see

the **Manually publishing a server** section.

The following is a list of servers that require you to always wait for the publishing interval to pass in order for the workbench to issue an automatic publish command.

- A remote WebSphere Application Server
- A local WebSphere Application Server when you are using the **Run server with resources on Server** publishing settings

The following is a list of servers that depends on the type of resource that is modified and the mode you are running the server determines if a wait is required for the publishing interval to pass in order for the workbench to issue an automatic publish command.

- A local WebSphere Application Server when you are using the **Run server with resources within the workspace** publishing settings

Mode of the server

In some cases, the requirement to wait for the automatic publishing interval to pass depends on the mode you are running the server. There are two modes of running applications on the server:

- Run on Server

- This command specifies to run your application on the server. It is available when you right-click your artifact in the Enterprise Explorer view and select **Run As > Run on Server**. The **Run on Server** wizard opens.

- Debug on Server

- This command specifies to control the running of your application by stopping at breakpoints, suspending threads, stepping through the code, and examining the contents of the variables while you are running your application on the server. It is available when you right-click your artifact in the Enterprise Explorer view and select **Debug As > Debug on Server**. The Debug on Server wizard opens.

Changes to JSP, HTML, graphic, and non-Java files

This subtopic is applicable to a local WebSphere Application Server by using the **Run server with resources within the workspace** publishing settings.

If you change a JSP file, HTML file, GIF file, JPG file, or similar resource, and save the file while the server is running, you must only refresh the web browser for the server to recognize the change. However, if the application contains multiple roots, you need to issue a publish command for the server to recognize the change. To determine whether the structure of your application contains a single or multiple roots use the Project Structure Validator. For details, see [Creating and configuring Java™ EE projects using wizards](#).

Changes to servlets and related classes

This subtopic is applicable to a local WebSphere Application Server by using the **Run server with resources within the workspace** publishing settings.

If you change a servlet and save the file while the server is running, the requirement to wait for the automatic publishing interval depends on the mode you are running the server.

- **Run on Server:** You need to wait for the automatic publishing interval to pass and then refresh the web browser for the server to recognize the change.
- **Debug on Server:** Hot method replace automatically runs in debug mode for WebSphere Application Server. You do not need to wait for the automatic publishing interval to pass as changes to the servlets are dynamically reloaded to the local server. However, if the application contains multiple roots, you need to issue a publish command for the server to recognize the change. To determine whether the structure of your application contains a single or multiple roots use the Project Structure Validator. For details, see [Creating and configuring Java EE projects using wizards](#).

Changes to a bean class of an EJB

This subtopic is applicable to a local WebSphere Application Server by using the **Run server with resources within the workspace** publishing settings.

If you change a bean class of an EJB and save the file while the server is running, the requirement to wait for the automatic publishing interval depends on the mode you are running the server.

- **Run on Server:** You need to wait for the automatic publishing interval to pass for the server to recognize the change.
- **Debug on Server:** Hot method replace automatically runs in debug mode for WebSphere Application Server. You do not need to wait for the automatic publishing interval to pass as changes to a bean class of an EJB are dynamically reloaded to the local server. However, if the application contains multiple roots, you need to issue a publish command for the server to recognize the change. To determine whether the structure of your application contains a single or multiple roots use the Project Structure Validator. For details, see [Creating and configuring Java EE projects using wizards](#) topic.

Changes to a local or remote interface of an EJB

This subtopic is applicable to a local WebSphere Application Server by using the **Run server with resources within the workspace** publishing settings.

If you change a local or remote interface of an EJB and save the file while the server is running, you must wait for the automatic publishing interval to pass.

Changes to resources within an enterprise application

This subtopic is applicable to a local WebSphere Application Server by using the **Run server with resources within the workspace** publishing settings.

Changes to resources within an enterprise application include the deployment descriptor files, and the Deployment page of the Application Deployment Descriptor editor. If you change any of these resources and save the file while the server is running, you must wait for the automatic publishing interval to pass.

*Table 1. Summary of wait requirements for the automatic publishing interval to pass when resources are modified while the server is running in different modes with the **Run server with resources within the workspace** publishing settings*

	Do you need to wait for the automatic publishing interval to pass?	
Resource modified	Run on Server	Debug on Server with Hot method replace
JSP	No, you need only to refresh the web browser for the server to recognize the change. ¹	
HTML	No, you need only to refresh the web browser for the server to recognize the change. ¹	
Servlets	Yes, and then refresh the web browser for the server to recognize the change.	No. ¹
EJB bean class	Yes.	No. ¹
EJB local and remote interface	Yes.	
Deployment descriptor files	Yes.	
Deployment page of the Application Deployment Descriptor editor	Yes.	

Note: Hot method replace automatically runs in debug mode for WebSphere Application Server.

Related concepts:

[Debugging with hot method replace for a WebSphere Application Server](#)

1 However, if the application contains multiple roots, you need to issue a publish command for the server to recognize the change. To determine whether the structure of your application contains a single or multiple roots use the Project Structure Validator. For details, see [Creating and configuring Java EE projects using wizards](#) topic.

Limitations and troubleshooting tips for Server tools

This file contains a comprehensive list of permanent limitations and troubleshooting tips for the Server tools.

Content

Important: Applicable edition: Full profile

The limitations and troubleshooting tips that you might encounter while you are working with the server are divided into the following sections.

- [General workbench issues](#)
- [Secured server or port connection](#)
- [Adding, removing or publishing applications to the server](#)
- [WebSphere® Application Server profile](#)
- [Using the Administrative Console](#)

General workbench issues

- [Troubleshooting if a problem is related to the workbench or the application server](#)
- [Limitations of servers due to invalid characters](#)
- [Limitations of server target validation](#)
- [Limitation of WebSphere Application Servers when workspace path begins with a backslash](#)
- [Limitation on the workbench to work with a WebSphere Application Server that changed its host name](#)
- [Limitations of servers due to long directory names](#)
- [The server might not stop completely when stopping the server from the Servers view](#)
- [A timeout error display when starting the server immediately after the server displays Stopped](#)
- [Limitations of X on Linux when using WebSphere Application Server](#)
- [The internal browser does not support IPv6 address](#)
- [Workspace migration limitation](#)
- [Web project wizard limitation](#)

Port connections or secured server

- [Server port in use when restarting development environment](#)
- [Limitations of desktop firewalls](#)
- [Long delays when establishing RMI connection after losing network connectivity](#)
- [The internal browser cannot display the administrative console for a secured WebSphere Application Server](#)
- [Administrative console login fails when running in secure mode](#)
- [Problems working with a secured server using SSL connections](#)
- [Unable to connect to a remote WebSphere Application Server hosted on a Red Hat Enterprise Linux machine](#)
- [Server connection or publishing problems when connecting to a remote WebSphere Application Server](#)
- [Unable to bind to ports on a Linux operating system after restarting the server](#)

Adding, removing, or publishing applications to the server

- [Troubleshooting if a problem is related to the workbench or the application server](#)
- [Limitation on Java 2 security application policy on WebSphere Application Servers](#)
- [Removing EJB module shared in multiple EAR projects](#)
- [Limitations of Microsoft SQL Server JDBC driver when publishing to a remote WebSphere Application Server](#)
- [Publishing fails with duplicate class error](#)

- Unable to use the workbench to publish the same application that was installed using the Administrative Console

WebSphere Application Server profile

- Resolving a profile name is invalid error message

Using the administrative console

- The internal browser cannot display the administrative console for a secured WebSphere Application Server
- Administrative console login fails when running in secure mode
- The internal browser does not support IPv6 address
- Unable to use the workbench to publish the same application that was installed using the Administrative Console
- Must restart server after changing the WebSphere Application Server configuration

- Invalid profile name

This topic describes how to resolve a `Profile name is invalid` error message.

- Universal Test Client: Internal browser does not support IPv6 address

If you specify an Internet Protocol version 6 (IPv6) address for the host name of the server, the internal browser might have problems displaying content, such as from the administrative console.

Parent topic: [Testing and publishing on a server](#)

Troubleshooting if a problem is related to the workbench or the application server

If you encounter errors or problems when you are running your application on the server by using the development workbench, try reproducing the problem by using only the tools that are provided by the application server without the use of the development workbench. For example, to reproduce the problem on WebSphere® Application Server, try installing and publishing the enterprise (EAR) application by using only the administrative console, see the **Installing enterprise application files with the console** topic available in the information center for WebSphere Application Server.

Important: Applicable edition: Full profile

If you can reproduce the error or problem by using only the tools of the application server without using the development workbench, the problem most likely is isolated to the application server. To diagnosis the problem further, see the publications and the support resources available for the application server.

If the problem is not reproducible by using only the tools of the application server, you can further diagnose the problem with the development workbench. You can try to resolve the problem by searching through the product knowledge bases at the IBM Support website.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Related information:

[IBM Support portal](#)

[Installing enterprise application files with the console](#)

Limitations of servers due to invalid characters

If you install this product into a directory whose name contains a dollar sign (\$) or any odd character such as #, %, +, or *, the server might not be created or might not start. To avoid an unsuccessful startup, do not install this product into a directory that contains any of these characters.

When you create a server or server project, do not include #, %, &, or * in the name. WebSphere® Application Server does not support these characters.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Limitations of server target validation

When you target a project to one server type and run on a different server type (which might have incompatibilities with each other), no errors appear during publish or startup. It is your responsibility to make sure that the targeted server and deployed server are matched in server type and version level.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Limitation of WebSphere Application Servers when workspace path begins with a backslash

WebSphere® servers might not start when you use a workspace that begins with a backslash, for example `\workspace` or `\myworkspaces\work1`

Important: Applicable edition: Full profile

If you already started the development environment with a workspace that begins with a backslash, follow these steps to allow the WebSphere servers to start:

1. Shut down the development environment.
2. Restart the development environment (use the `-setworkspace` flag if you previously chose to hide the workspace selection dialog box on startup).
3. When prompted for the workspace location, add the drive letter to the beginning of the workspace path. For example, `\workspace1` would become `C:\workspace1`
4. You can now start your existing WebSphere Application Server.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Limitation on the workbench to work with a WebSphere Application Server that changed its host name

Important: Applicable edition: Full profile

After a change in the host name of the server, this product might fail to work properly with the server in any of the following manners:

- If you start the server by using the workbench, the server status in the Servers view does not go beyond `Starting`. However, the server is started.
- If you start the server outside the development environment, for example by using `startServer.bat` (in Linux, `startServer.sh`) or `wsadmin` script, the workbench cannot connect to the server and the server status in the Servers view is `Stopped`.
- If the server is started and the server status in the Servers view is `Started`, the output in the Console view is not available.

When the host name of the server is changed, update this change in the server configuration files.

To update server configuration files with the changes in the host name of the server:

1. On the machine where the host name is changed, start the server.
2. After the server is started, open the administrative console.
 - A. Open a web browser with the following address: `http://<machine_name>:<WC_adminhost>/ibm/console`, where the *machine_name* is the host name or IP address of the machine where the server is running and *WC_adminhost* is the value of `WC_adminhost` port. You can find the value of the `WC_adminhost` port in the `serverindex.xml` file available in the `<WAS_install_directory>\profiles\<profileName>\config\cells\<cellName>\nodes\<nodeName>` directory, where *WAS_install_directory* is the installation directory of the WebSphere® Application Server.
 - B. Log in to the administrative console.
3. On the administrative console, select **Server > Application servers**.
4. On the administrative console, verify that the **Configuration** tab is open.
5. Under the **Communications** section, select the **Ports** link.
6. The **Host** column shows the current host name setting for the server.
7. Change all occurrences of the old machine name to the new fully qualified host name (or 127.0.0.1). **Tip:** If you use the IP address, 127.0.0.1, instead of the fully qualified host name, some functions on the server might not function; for example, when you try to do a JNDI lookup from a remote machine by using this server.
 - A. Under the **Port name** column, click the *port_name* link that requires an update of the host name.
 - B. In the **Host** field, specify the new fully qualified host name.
 - C. Apply and save your changes in the administrative console.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Limitations of servers due to long directory names

If you use a workspace in a directory with a long path or choose long names for your Enterprise Application project or web project, you might get errors when you start a server, or when testing files on a server.

Important: Applicable edition: Full profile

For example, the following shows what a WebSphere® Application Server error might look like: `Error Message: JSPG0113E:`

JSP file

```
"Xxx/Yyy_jsp_0.java (Filename too long)" not found
```

If you receive this error, you might take any of the following actions:

- Move your workspace to a location with a shorter path, for example `C:/workspace`.
- Rename your Enterprise Application project or other projects with a shorter name.
- Decrease the folder depth of the file within the application. For example, move the JSP pages into a common folder or root of the web content folder instead of nesting them further down.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

The server might not stop completely when stopping the server from the Servers view

When you stop the server from the Servers View, the server might not completely stop. The Servers View displays as `stopped` but the server process might be in a non-responsive state. The non-responsive state usually occurs when artifacts such as your application or the workbench remains holding onto references to classes on the server. The following are example scenarios:

Important: Applicable edition: Full profile

- Applications that are in endless loops, or application hold on references to some classes on the server
- Applications that make Cloudscape or Derby database connection without cleaning up their connection
- Using the **Test Connection** button in the New Connection wizard of the data tools to open a connection to a Cloudscape or Derby database without disconnecting from the database

Restriction: Multiple connections to a single Cloudscape or Derby database are not supported due to a Cloudscape or Derby configuration restriction. If you maintain the database connection to the database from the Database Explorer and a server tries to make another Cloudscape or Derby connection through a data source, the second connection is going to fail. As a result, you need to close the connection from the Database Explorer before a server can establish a connection to the Cloudscape or Derby database.

To work around the problem, you to need use the functions from your operating system to stop the `Java™` process on which the server is running. Alternatively, you can restart the workbench to force the reference to be released. The last example scenario that is described in the third bullet, you can use the Database Explorer view to connect and then disconnect from the Cloudscape or Derby database connection.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

A timeout error displays when starting the server immediately after a stop request is issued

Starting a WebSphere® Application Server immediately after a stop request is issued can result in a timeout error message; even if the status of the server in the Servers view displayed `Stopped` before you starting the server.

Important: Applicable edition: Full profile

Here is the text of the timeout error message: `Server <server_name> was unable to start within <nnn> seconds. If the server requires more time, try increasing the timeout in the server editor.`

Where `<server_name>` is the name of the server in the Servers view, and `<nnn>` is the number of seconds specified in the **Start** control under the **Timeouts** section of the server editor. Although the status of the server in the Servers view displays as `Stopped`, the server might not actually be stopped. If the server is on a slow processing machine or has a slow response time, more time is required to stop the server.

If the timeout error message displays, click **OK** and the server stops and starts completely. If you find the server is stopped and the Servers view does not reflect the same `Stopped` status, restart the development workbench.

However, when you want to restart a server that is already running and started, right-click the server in the Servers view and select **Restart**; instead of selecting **Stop** and immediately followed by a **Start** command. Selecting the **Restart** option gives control to the workbench to handle the changeover between the stop and start of the server.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Related tasks:

[Stopping a server](#)

[Starting a server](#)

Limitations of X on Linux when you are using the WebSphere Application Server

It is a known problem that when you use the X GUI on Linux when you are using WebSphere® Application Server, you might get the following error, which prevents the **Run** dialog box from appearing:

Important: Applicable edition: Full profile

Caused by: java.lang.InternalError: Can't connect to X11 window server using ':0.0' as the value of the DISPLAY variable.

To work around this problem, set the system environment variable `JAVA_MMAP_MAXSIZE` to a value less than 20. The default is `JAVA_MMAP_MAXSIZE=20`. Any JAR file that is less than 20 megabytes in size is memory that is mapped. JAR files are mapped into memory if the JVM considers them small, which allocates a file descriptor for each JAR file. Because WebSphere Application Server use many JAR files, if you reduce the maximum size of the JAR files that are memory that is mapped, fewer file descriptors are allocated, and the X GUI accesses the JAR files successfully.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

A runtime problem occurs when a Web 2.4 project requests EJB 3.0 injections

If a servlet is contained in a web 2.4 project and is requesting EJB 3.0 injections, a runtime error in WebSphere® Application Server v7.0 results. An example of a runtime error that can be produced is a null pointer error that is produced from a doGet or doPost method.

Important: Applicable edition: Full profile

Select one of the following solutions:

- Move the servlet into a web 2.5 project
- Edit the MANIFEST.MF file with the source editor or the workbench's text editor, and add the following line of code.

```
UseEJB61FEPScanPolicy: true
```

In the Enterprise Explorer, you can find this MANIFEST.MF file in the Web 2.4 project under the *<web project>*\WebContent\META-INF directory.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Universal Test Client: Internal browser does not support IPv6 address

If you specify an Internet Protocol version 6 (IPv6) address for the host name of the server, the internal browser might have problems displaying content, such as from the administrative console.

Important: Applicable edition: Full profile

Typically, the notation of an IPv6 address is written as eight groups of four hexadecimal digits, where each group is separated by a colon (:). For example: 2001:0db8:85a3:08d3:1319:8a2e:0370:7348 Here are examples where you can specify an IPv6 address as the host name of the server:

- In a **Server's host name** field, for example in the Servers view, right-click and select **New**. In the Define a New Server page, the **Server's host name** field is available.
- In a **host name** field, for example in the Server view, right-click a server and select **Open**. Under the General information section, there is a **host name** field available.

Select one of the following workarounds to resolve this problem.

- Use an external web browser that supports IPv6 addresses. Windows Internet Explorer 7, Mozilla Firefox 2.0 or later is known to support IPv6 addresses, however, there are known problems using Internet Explorer 6 to display content by using IPv6 addresses. You can configure the workbench to use an external web browser by using the web Browser preference page (in the toolbar select **Window > Preference > General > Web Browser**)
- If you want to continue to use the internal browser, specify the host name instead of the IPv6 address to the server.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Workspace migration limitation

If you have a WebSphere® Application Server V8.5 run time that is installed in your workspace that uses Java™ 6, and you import a project that was created with a WebSphere Application Server V8.5 run time that uses Java 7, the migration wizard fails because the installed WebSphere Application Server V8.5 run time does not support Java version 7.

Workspace migration limitation scenario

Important: Applicable edition: Full profile

- In your workspace, you have a WebSphere Application Server V8.5 runtime installed that uses Java 6.
- Into this same workspace, you import a project that is created with WebSphere Application Server v8.5 run time that uses Java 7.
- The migration wizard appears, but there is no target run time to select, because the Java facet version of the imported project is 1.7, which is not supported by your installed WebSphere Application Server V8.5 run time.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Web project wizard limitation

If you install a WebSphere® Application Server V8.5 run time in your workspace that uses Java™ 6, and then change the JRE version of your WebSphere Application Server V8.5 run time to v 1.7. after you create a web project, when you create a second web project, it will use the JRE 1.6 facet.

Web project wizard limitation scenario

Important: Applicable edition: Full profile

- You install a WebSphere Application Server V8.5 run time in your workspace that uses Java 6.
- You create a web project that targets this WebSphere Application Server V8.5 run time.
- In the runtime editor, you change WebSphere Application Server V8.5 run time to target JRE V1.7,
- Then, you create another web project targeting the same WebSphere Application Server V8.5 run time, this second web project uses JRE version 1.6. instead of 1.7 because the web project wizard caches the facet from the previous creation.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Server port in use when you are restarting the development environment

If the product crashes or the Java™ product process is stopped externally, such as in the Windows Task Manager, while the server is running, you might receive a `Server port port_number is in use` message when you try to start the server after you restart the development environment. Some server resources are not fully released.

Important: Applicable edition: Full profile

The work-around is to stop the server's Java process before you start the server. To determine which Java process to stop, open the process identification file for the server at `profile_path/logs/server_name/server_name.pid`. The number in the `.pid` file identifies which Java process to stop.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Limitations of desktop firewalls

If you are running a desktop firewall that blocks access to your systems ports, you might receive warnings or errors when you are starting servers, or the servers might fail to start. To work around the problem, you must give TCP/IP port access to the server process and use a SOAP connection.

Important: Applicable edition: Full profile

If there is a firewall between the development environment and the server, use the SOAP connector port rather than the RMI (ORB bootstrap port). The SOAP connector port is more firewall compatible. The default setting of the SOAP port is port 8880 and selected when you are working with a remote server. For more information, see [Setting the connection to the WebSphere® Application Server](#).

To give TCP/IP port access, add a rule that allows a particular process access to a number or range of ports.

- For WebSphere Application Server, the Java™ process that is used to start the server can be found at: **Windows** *x*:

```
/runtimes/server_name/java/jre/bin/javaw.exe
```

```
Linux x:/runtimes/server_name/java/jre/bin/javaw
```

Where *x* is the installation directory of WebSphere Application Server.

- For the TCP/IP Monitor, the monitor is started within the development environment process.

- If you are debugging on the server, a dynamic port number is used by default. If you want to restrict the debugger to use a particular port, you must edit the server and specify a debug port. This is done by using the administrative console by changing the address parameter of the Debug Argument under the Java virtual machine settings. For more information, see the Java virtual machine topic available in the WebSphere Application Server information Center:

- For WebSphere Application Server V8.5, see the [Java virtual machine settings](#) link

- For WebSphere Application Server V8.0, see the [Java virtual machine settings](#) link

- For WebSphere Application Server V7.0, see the [Java virtual machine settings](#) link

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Long delays when establishing RMI connection after losing network connectivity

On Windows operating systems, if you are using the remote method invocation (RMI) port to connect to your WebSphere® Application Server, you might experience long delays to establish a connection to the server after you lose network connectivity. This can occur even if the server is local and the network connectivity is lost only temporarily, which is common in a wireless network environment. If you know that the server is started, but the status in the Servers views displays `Stopped` or `Started`, try to see if you can establish a connection to the server by switching the server connection from RMI to SOAP. The status of the server changes to `Started`.

Important: Applicable edition: Full profile

You have a couple of options that are available to establish connection to a server in a wireless network environment:

- The easiest and safest option, is to switch your connection to use the SOAP port. SOAP connections that lose network connectivity can recover more quicker compared to an RMI connection.
- If you must use an RMI connection, you can try to modify the default settings that pertain to the Domain Name System (DNS) caching on the Windows operating system. For details, refer to the following Microsoft support article. <http://support.microsoft.com/kb/318803> Windows operating system has a built-in DNS caching that maintains resolved host names. Built-in DNS caching allows for faster DNS lookups. However, there is a disadvantage to having faster DNS lookups, which is if a DNS lookup fails. The Windows operating system caches the failed value, for a default time of 300 seconds. So even if the DNS server can resolve the lookup shortly thereafter, it does not actually attempt the lookup until the cache time expires. As a result, a failed DNS lookup with default settings can take as many as 5 minutes before a lookup is attempted again. Setting the cache time to 0 seconds forces the Windows operating system to never cache failed DNS lookup queries, and allow the reconnection to occur as soon as the DNS becomes available.

The following is an example of disabling DNS caching for failed lookups on Windows operating systems:

In the following registry key: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Dnscache\Parameters`

Add one of the following registry values:

- For Windows XP or 2003: `"MaxNegativeCacheTtl"=dword:00000000`
- For Windows 2000: `"NegativeCacheTime"=dword:00000000`

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

The internal browser cannot display the administrative console for a secured WebSphere Application Server

On a Linux Motif platform, a blank web page might display on an internal browser when you run the administrative console for a secured WebSphere® Application Server. The internal web browser is the default set browser that comes with the workbench.

Important: Applicable edition: Full profile

Under some circumstances, the internal web browser fails to serve pages when the application server target is running in secure mode. To run the administrative console on a secured server, choose one of the following options.

- Use an external web browser instead of an internal web browser:

1. Open an external web browser.

2. In the address field, type `http://<hostname>:<WC_adminhost>/ibm/console`, where `<hostname>` is the fully qualified DNS name or IP address of the host machine where your server is started and `<WC_adminhost>` is the port to communicate to the administrative console of the WebSphere Application Server. For example, `http://localhost:9060/ibm/console`**Tip:** You can determine the `<WC_adminhost>` port value of a configured profile by referencing its server configuration files. The `<WC_adminhost>` port value is stored in the `serverindex.xml` file that is in the following directory: `x:\profiles\<profileName>\config\cells\<cellName>\nodes\<nodeName>`, where `x` is the directory WebSphere Application Server is installed.

- Verify that the workbench is using HTTP instead of HTTPS, by clearing the **Use HTTPS when running server resources** check box. See *Changing the hypertext transfer protocol settings for WebSphere Application Server* topic.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Related tasks:

[Changing the hypertext transfer protocol settings for WebSphere Application Server](#)

Administrative console login fails when running in secure mode

Important: Applicable edition: Full profile

If you are running a WebSphere® Application Server in secure mode, you might not be able to log on to the administrative console and receive the following message.

Unable to process login. Check user ID and password and try again.

This might occur even though the server was successfully started in secure mode by using the same user ID and password. This behavior is intermittent, and appears to be associated only with Microsoft Internet Explorer. To work around this problem, complete one of the following steps:

- From the main menu, select **Preferences > General > Web Browsers**. Select the **Use external Web browser** radio button and add Microsoft Internet Explorer as an external web browser by clicking the **New** button. In the **Location** field, browse or type the external directory where Microsoft Internet Explorer is installed and click **OK**.
- If you are using Microsoft Internet Explorer externally fails as well, try first to clear the browser cache by clearing the temporary internet files.
- If you still cannot log in to the administrative console with the external Internet Explorer, and you continue to get the message, select **Preferences > General > Web Browser** and add Mozilla, Netscape or Opera as an external web browser by clicking the **New** button. In the **Location** field, browse or type the external directory where Microsoft Internet Explorer is installed and click **OK**. When you are finished using the administrative console while the server is running in secure mode, you can reset your previous preferences.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Unable to connect to a remote WebSphere Application Server hosted on a Red Hat Enterprise Linux machine

This topic discusses the steps that you must configure on a Red Hat Enterprise Linux 5 to set up the host name for accessing the WebSphere® Application Server from a remote machine.

Important: Applicable edition: Full profile

If you are hosting a WebSphere Application Server on a Red Hat Enterprise Linux machine, complete the following steps for configuring the host name to allow access from remote connections:

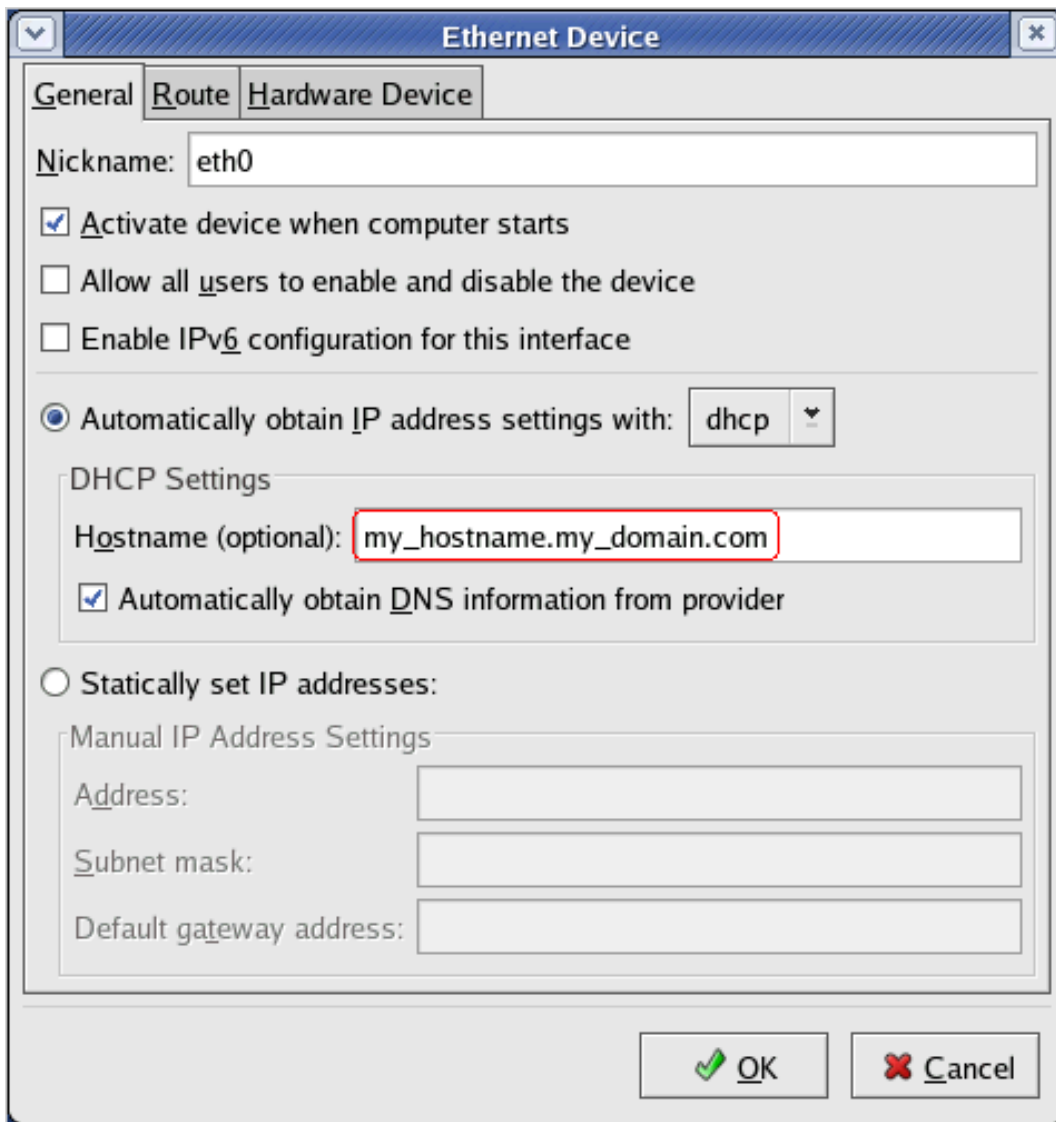
1. In the Red Hat Enterprise Linux machine, open the `hosts` file available in the `/etc` directory. Verify that the host name of your machine is not specified in the `127.0.0.1` entry. If the host name of your machine is specified in the `127.0.0.1` entry, change the host name to `localhost.localdomain`. Your `127.0.0.1` entry in your `hosts` file looks like:

```
# Do not remove the following line, or various programs
```

```
# that require network functionality will fail.
```

```
127.0.0.1      localhost.localdomain  localhost
```

2. Verify that the host name of your machine is specified in the DHCP settings.
 - A. Open the Network Configuration dialog box. By default in a Red Hat Enterprise Linux operating system, you can find this dialog by selecting **System > Administration > Network**. The Network Configuration dialog opens.
 - B. In the Devices page, highlight your Ethernet entry. In most systems, the device name is `eth0`.
 - C. Select the **Edit** button from the toolbar. The Ethernet Device dialog opens.
 - D. Use the **Hostname** field under the **DHCP Settings** to specify the host name of your machine.



3. Choose the appropriate task to activate your changes:

- Restart the network services
- Restart the computer

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Server connection or publishing problems when connecting to a remote WebSphere Application Server



Important: Applicable edition: Full profile

An incorrect host name that is specified to a remote WebSphere® Application Server can cause unexpected behavior from the workbench:

- When you use the remote method invocation (RMI) port to connect to the server, the status of the server displays `Stopped` in the Servers view; given the server is started.
- When you are using the SOAP port to connect the server, the status of the server displays correctly in the Servers view; however when you attempt to publish, the server becomes unresponsive.

To troubleshoot and determine that the host name is configured correctly, complete the following steps:

1. Verify that the correct host name is specified by matching the values that are specified in the server editor and the `serverindex.xml` configuration file.
 - A. In the Servers view, right-click the server and select **Open**. The server editor opens.
 - B. Under the **General Information** section, verify the value that is specified in the **Host name** field matches the value that is specified for the `hostName` parameter in the `serverindex` tag that is defined in the `serverindex.xml` file. The `serverindex.xml` file can be found on the remote machine that is running the WebSphere Application Server in the following directory: `<WAS_install_directory>\profiles\<profileName>\config\cells\<cellName>\nodes\<nodeName>`, where `WAS_install_directory` is the installation directory of the remote WebSphere Application Server.

Tip: If you notice that the actual host name of the remote machine is different from the host name that is specified in the `serverindex.xml` file, see the [Limitation on the workbench to work with a WebSphere Application Server that changed its host name](#) topic for details on updating the host name in the server configuration files.
2. If the Domain Name System (DNS) is not available or you see problems when you resolve host names into IP addresses, try configuring the `hosts` file from the operating system that runs the workbench. In the `hosts` file, add a mapping of the IP address and host name of the remote machine that runs the server. The location of the `hosts` file in the file system varies between operating systems. See the publications of the operating system for information on where to find the `hosts` file and how to add the mappings.
3. Verify that there is a connection between the local and remote machine by testing the connection with a ping request.
 - A. In the local machine that is running the workbench:
 -  Open a command prompt
 -  Open a terminal window
 - B. Type: `ping <host name>`

Where `host name` is the name of the remote computer that is running the server:
 - C. The ping command displays whether there is a reply from the host and how long it takes to receive a reply. If there are network connection problems, the response from the ping command returns an error message, for example, `Ping request could not find host <hostname>. Please check the name and try again.`

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Unable to bind to ports on a Linux operating system after restarting the server

Important: Applicable edition: Full profile

You might experience a long wait period for the Linux operating systems to release the TCP ports after you restart a WebSphere® Application Server, in any of the following scenarios.

- Start the server immediately after you stop the Java™ process
- Restart the server
- Restart the server in debug mode

The operating system can take more than 3 minutes to release the port. As soon as the JVM receives the stop signal, information is passed to the operating system to delete all the listening ports. The operating system is responsible to clean up the ports that are in use. The following message might appear in the `SystemOut.log` file when the `startServer` command is issued immediately after you stop the server process. [6/26/11 4:41:59:647 EDT] 00000018 TCPPort E

```
TCPC0003E: TCP
```

```
Channel TCP_2 initialization failed. The socket bind failed for localhost and port 9083. The port may already be in use.
```

After the port is released by the operating system, WebSphere Application Server can bind to the port. You might see the following message after a few minutes: [6/26/11 4:44:04:677 EDT] 00000018 TCPChannel A TCPC0001I: TCP Channel TCP_2 is

```
listening on localhost port 9083.
```

There is no solution for this limitation. However, you can try to tune up the timeout value by logging in to the Linux operating system as a `root` user. After you are logged in, add the following entry in `sysctl.conf` file, typically available in the `etc` directory. `net.ipv4.tcp_fin_timeout = 30`

Adding a timeout entry in `sysctl.conf` file does not resolve the problem. However, the operating system is scheduled to clean up the ports within the specified time duration. In this case, the cleanup happens in 30 seconds.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Limitation on Java 2 security application policy on WebSphere Application Servers

The `{application}` policy in the Java™ 2 policy security files (for example `was.policy`) applies only to the EAR project.

Important: Applicable edition: Full profile

If you are using publishing to a WebSphere® Application Server with **Run server with resources within the workspace** enabled, the policy does not affect the modules that are contained in the EAR, for example EJB, Web, or connector module. The security policy of the modules is set by using the individual modules policies, for example, by using `{ejbComponent}`, `{webComponent}`, and `{connectorComponent}`.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Removal of an EJB module shared in multiple EAR projects

If an Enterprise JavaBeans (EJB) module is shared among multiple enterprise application (EAR) projects that run on a WebSphere® Application Server and one of the EAR projects are removed from the server, other EAR projects must be restarted before they can access the resources, such as bean classes, in the EJB project.

Important: Applicable edition: Full profile

If you do not do restart other EAR projects, you might see error messages similar to what is shown in the sample message. These errors happen because Java™ Naming and Directory Interface (JNDI) name in the EJB project are removed from the server when the EAR is removed.

Here is a sample error message:

```
00000028 SystemOut      O javax.naming.NameNotFoundException: Context: myCell/nodes/myNode/servers/server1, name:
ejb/ejbs/Session20Home: First component in name Session20Home not found. [Root exception is
org.omg.CosNaming.NamingContextPackage.NotFound: IDL:omg.org/CosNaming/NamingContext/NotFound:1.0]

    at com.ibm.ws.naming.jndicos.CNContextImpl.processNotFoundException(CNContextImpl.java:4730)
    at com.ibm.ws.naming.jndicos.CNContextImpl.doLookup(CNContextImpl.java:1907)
    at com.ibm.ws.naming.jndicos.CNContextImpl.doLookup(CNContextImpl.java:1862)
    at com.ibm.ws.naming.jndicos.CNContextImpl.lookupExt(CNContextImpl.java:1552)
    at com.ibm.ws.naming.jndicos.CNContextImpl.lookup(CNContextImpl.java:1354)
    at com.ibm.ws.naming.util.WsnInitCtx.lookup(WsnInitCtx.java:172)
    at javax.naming.InitialContext.lookup(InitialContext.java:363)
    at com.ibm.ivj.ejb.runtime.AbstractAccessBean.lookupAndCacheHome(AbstractAccessBean.java:224)
    at com.ibm.ivj.ejb.runtime.AbstractAccessBean.getGlobalHome(AbstractAccessBean.java:216)
    at com.ibm.ivj.ejb.runtime.AbstractAccessBean.getHome(AbstractAccessBean.java:249)
    at ejbs.Session20AccessBean.ejbHome(Session20AccessBean.java:50)
    at ejbs.Session20AccessBean.instantiateEJB(Session20AccessBean.java:80)
```

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Changing a Web or EJB project results in an application to be restarted with option Never publish automatically

When the publishing option **Never publish automatically** is enabled with the publishing setting for WebSphere Application Server **Run server with resources within the workspace** enabled, the WAR can still be restarted even without a publishing taking place. This problem is caused by automatic class reloading in WebSphere Application Server.

For example, if a new method is added to the servlet, the WAR will be restarted automatically even though the server is in a Republish state. To work around this known limitation:

1. Restart the server in debug mode. While in debug mode, the changes will take place, but the WAR will not be restarted since the changes are picked up as part of the hot code replace.
2. Disable WebSphere Application Server class reloading by specifying a WebSphere Extension Deployment Descriptor.
 - Select **web project > Java EE > Generate WebSphere Extensions Deployment Descriptor**.
 - The generated `ibm-web-ext.xml` file will be in the web project's WEB-INF.
 - Open `ibm-web-ext.xml` and go to the **Design** tab.
 - Uncheck **Enable Reloading**.
 - Republish your application
3. Open the server editor and select **Run server with resources on Server** from the **Publishing settings for WebSphere Application Server** section.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Limitations of Microsoft SQL Server JDBC driver when publishing to a remote WebSphere Application Server

It is a known problem that when you are publishing to a remote WebSphere® Application Server by using a Microsoft SQL JDBC driver, you might get the following error. This error prevents you from making a database connection:

Important: Applicable edition: Full profile

SystemOut

```
O [Microsoft][SQLServer 2000 Driver for JDBC]Error opening/loading com.microsoft.util.transliteration.properties.
```

SystemErr

```
R java.sql.SQLException: [Microsoft][SQLServer 2000 Driver for JDBC]Error opening/loading  
com.microsoft.util.transliteration.properties.
```

To work around this problem:

1. Copy the following Microsoft SQL Server JAR files to the *x:\AppServer\lib* folder (where *x* is the directory where WebSphere Application Server is installed:

```
- msbase.jar  
- mssqlserver.jar  
- msutil.jar
```

2. If you want the JAR files located elsewhere:

A. In the WebSphere Application Server installed directory, open the following directory. *x:\profiles\properties*

Where *x:\profiles* is the directory of your profile for the WebSphere Application Server.

B. Edit the server.policy file and change the permissions for the JAR files to read access, for example.// WebSphere

```
system classes  
  
grant codeBase "file:${was.install.root}/lib-" {  
    permission java.security.AllPermission;  
    permission java.io.FilePermission  
    "${was.install.root}/${lib$}/msbase.jar", "read";  
    permission java.io.FilePermission  
    "${was.install.root}/${lib$}/msutil.jar", "read";  
    permission java.io.FilePermission  
    "${was.install.root}/${lib$}/mssqlserver.jar", "read";  
};
```

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Publishing fails with duplicate class error

When you are publishing an application to a remote or local WebSphere® Application Server running with the **Run server with resources on Server** publishing setting, you might encounter the following error message:

Important: Applicable edition: Full profile

The publish encountered some problems and the application may not have been installed or it may have been successfully installed but was unable to start.

duplicate entry: history.jar/com/ibm/ejs/container/_EJSWrapper_Stub.class

You can encounter this problem when you import a Java™ EE archive that has class files without corresponding source files. During import, the workbench is going to place these class files into an `ImportedClasses` folder in the project. If you later add the source files to the project, the class files become generated in the output folder for the project. When you publish the project, you can receive an update request for the same class file twice. One update is in the `ImportedClasses` folder and another in the `output` folder of the project, which results in the duplicate entry error. You can work around this limitation by removing the class files that are referenced in the error message from the `ImportedClasses` folder. If these class files are generated for an EJB, you can safely remove all the generated class files from the `ImportedClasses` folder.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Related tasks:

[Publishing settings for a WebSphere Application Server](#)

Unable to use the workbench to publish the same application that was installed by using the administrative console

An application that is published by using the workbench might display a different application name on the server, as the same application installed by using the administrative console from WebSphere® Application Server.

Important: Applicable edition: Full profile

This scenario occurs when an application contains different names for the Enterprise Application (EAR) project and the name that is specified in the `display-name` tag in the application deployment descriptor (`application.xml`) file. The development workbench uses the EAR project name as the application name when published onto the server; whereas, the administrative console uses the `display-name` of the application that is contained in the `application.xml` file. If you install this application by using the administrative console and then by using the workbench to publish this same application onto the same server, the application is going to exist twice on the server under two different names: One of the application names is generated when you install the application by using the administrative console and corresponds to the `display-name` from the `application.xml` file. The other application name is generated when you publish the application by using the workbench and corresponds to the EAR project name. Using the workbench to make more updates and publish to this application fail and result in the following runtime error message followed by various publishing exceptions: `The publish encountered some problems and the application may not have been installed or it may have been successfully installed but was unable to start.`

Here are examples of publishing exceptions in the Console view of the workbench:

```
- javax.naming.NameAlreadyBoundException
- com.ibm.ws.webcontainer.exception.WebAppNotLoadedException
```

If you want flexibility in updating and publishing the same application by using both the workbench or the administrative console, you need to match the EAR project name with the `display-name` of the application that is contained in the `application.xml` file. When you create an EAR project by using the workbench, by default the workbench uses the name that you provide for the EAR project to automatically generate the corresponding `display-name` of the application that is contained in the `application.xml` file.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Invalid profile name

This topic describes how to resolve a `Profile name is invalid` error message.

Important: Applicable edition: Full profile

Here is the error message that displays in the Console view: `Profile name <profileName> is invalid. Specify a profile name that exists on the server.`

There are two known scenarios that you can encounter this error message:

- When you migrate a server project that contains a server with a WebSphere® Application Server profile name that does not exist on the target server, you might encounter the `Profile name is invalid` error message when you try to publish an application on the target server, such as when you use the **Run on Server** option.
- When the **Profile name** field is empty and you attempt to start the server.

Symptom

In the server editor, under the **Server** section the **Profile name** drop-down list contains an empty entry or becomes disabled and you cannot select another profile. You can open the server editor by right-clicking the WebSphere Application Server entry in the Servers view and select **Open**.

Resolving the problem
Select a valid and existing profile from the **Profile name** drop-down list. If the **Profile name** drop-down list is empty, you must define a profile; see *Creating a profile for a WebSphere Application Server* topic in the related reference section for details.

If you are unable to edit the **Profile name** field, verify that there are projects added to the server. When a project is added to the server, the **Profile name** field cannot be changed. Complete the following steps to modify the **Profile name** field in the server editor:

1. In the Servers view, right-click the WebSphere Application Server entry and select **Add and remove projects**.
2. Select the **Remove All** button.
3. Click **Finish** in the **Add and remove projects** wizard and allow for the changes to occur on the server.
4. Open the server editor by right-clicking the WebSphere Application Server entry in the Servers view and select **Open**.
5. Under the **Server** section, you can now select a valid and existing profile from the **Profile name** drop-down list.
6. Save the changes in the server editor by typing `Ctrl + s`.
7. In the Servers view, right-click the WebSphere Application Server entry and select **Add and remove projects**.
8. Click **Add All > Finish**.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)

Related reference:

[Profile creation for a WebSphere Application Server](#)

Must restart server after changing the WebSphere Application Server configuration

The workbench might be unable to detect server states and run modes when certain configuration changes are made while the server is running.

Important: Applicable edition: Full profile

If you modify the WebSphere® Application Server configuration, for example the LOG_ROOT variable or the debugMode field, by using the WebSphere Application Server administrative console or modify the corresponding files directly when the server is running, you might need to restart the server from the command line in order for the workbench to correctly detect server states and the server mode, such as debugging.

Parent topic: [Limitations and troubleshooting tips for Server tools](#)