



Administering applications and their environment

Note

Before using this information, be sure to read the general information under “Notices” on page 519.

Compilation date: September 11, 2008

© Copyright International Business Machines Corporation 2008.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

How to send your comments	ix
Changes to serve you more quickly	xi
Chapter 1. Overview and new features for administering applications and their environments	1
Web applications	1
EJB applications	1
Client modules	3
Web services	3
Data access resources	6
Messaging resources	7
Mail, URLs, and other J2EE resources	8
Security	8
Introduction: System administration	17
Introduction: Administrative console	18
Introduction: Administrative scripting (wsadmin)	19
Introduction: Administrative commands	19
Introduction: Administrative programs	20
Introduction: Administrative configuration data	20
Product library, directories, subsystem, job queue, job description, and output queues	20
Introduction: Application servers	22
Introduction: Application servers	22
Introduction: Web servers	24
Introduction: Environment	25
What is new for scripted administration (wsadmin)	25
Introduction: Variables	27
Language versions offered by this product	28
Chapter 2. Using the administrative clients	31
Using the administrative console	31
Installing and uninstalling the administrative console	31
Starting and logging off the administrative console	32
Administrative console buttons	35
Administrative console page features	38
Console layout	39
Console navigation	40
Administrative console browser support	42
Console accessibility	42
Welcome	44
My tasks.	44
Specifying console preferences	44
Accessing help and product information from the administrative console	49
Administrative console: Resources for learning.	74
Console identity	75
Changing the console session expiration	76
Changing the class loader order of the console module deployed in Integrated Solutions Console	76
Console identity string.	77
Developing console modules	79
Overview of Integrated Solutions Console	80
Example: Console module samples	82
Setting up the development environment	83
Developing your first console module	83
Adding advanced API features.	92

Console module schemas	98
Using administrative programs (JMX)	112
Java Management Extensions	114
Creating a custom Java administrative client program using WebSphere Application Server administrative Java APIs	116
Creating a Java Management Extensions client program using the Java Management Extensions Remote application programming interface	125
Extending the WebSphere Application Server administrative system with custom MBeans	129
Administrative programs for multiple Java Platform, Enterprise Edition application servers	142
Deploying and managing a custom Java administrative client program with multiple Java Platform, Enterprise Edition application servers	143
Java Management Extensions V1.0 to Java Management Extensions V1.2 migration	145
Java Management Extensions interoperability	145
Using command line tools	147
Configuring Qshell to run WebSphere Application Server scripts	148
Qshell environment variables	149
Example: Security and the command line tools	149
manageprofiles command	149
startServer command	162
stopServer command	164
stopNode command	166
serverStatus command	168
cleanupNode command	169
backupConfig command	169
restoreConfig command	171
EARExpander command	173
GenPluginCfg command	174
versionInfo command	176
genVersionReport command	179
historyInfo command	181
genHistoryReport command	183
Granting authority to a profile using the i5/OS command line	184
Revoking authority to a profile using the i5/OS command line	185
checkprereqs command	185
chgwassvr command	186
configureOs400WebServerDefinition command	189
dspwasinst command	191
enablejvm command	191
enbprfwasm command	193
heapMonitor command	193
ivt command	194
port validator tool	195
prerequisite validator tool	196
removeOs400WebServerDefinition command	197
rvkwasaut command	198
servicetools command	199
updwashost command	200
grtwasaut command	200
registerNode command	202
deregisterNode command	203
revokeCertificate command	205
requestCertificate command	206
createCertRequest command	208
queryCertificate command	210
Chapter 3. Using Ant to automate tasks	213

Chapter 4. Starting and stopping quick reference	215
Chapter 5. Backing up and recovering the application serving environment.	217
Chapter 6. Class loading	219
Class loaders	219
Configuring class loaders of a server	223
Class loader collection	225
Class loader ID	225
Class loader order	225
Class loader settings	225
Configuring application class loaders	226
Configuring Web module class loaders	227
Class loading: Resources for learning	229
Chapter 7. Deploying and administering enterprise applications	231
Enterprise (Java EE) applications	231
System applications	232
Common deployment framework	232
Installing enterprise application files	233
Installable enterprise module versions	234
Ways to install enterprise applications or modules	235
Installing enterprise application files with the console	238
Example: Installing an EAR file using the default bindings	262
Example: Installing a Web Services Sample with the console	263
Installing enterprise modules with JSR-88	265
Customizing modules using DConfigBeans	266
Enterprise application collection	267
Name	268
Application Status	268
Startup order	269
Enterprise application settings	269
Configuring enterprise application files	270
Application bindings	271
Configuring application startup	276
Configuring binary location and use	278
Configuring the use of class loaders by an application	283
Manage modules settings	287
Mapping modules to servers	289
Mapping virtual hosts for Web modules	290
Mapping properties for a custom login or trusted connection configuration	292
Viewing deployment descriptors	293
Metadata for module settings	295
Starting or stopping enterprise applications	296
Disabling automatic starting of applications	297
Target specific application status	298
Exporting enterprise applications	300
Exporting enterprise application files	301
Exporting DDL files	302
Updating enterprise application files	302
Ways to update enterprise application files	303
Updating enterprise applications with the console	305
Preparing for application update settings	306
Hot deployment and dynamic reloading	310
Uninstalling enterprise applications	318
Removing enterprise files	318

Deploying and administering applications: Resources for learning	319
Chapter 8. Managing applications through programming	321
Application management	322
Accessing the application management function.	323
Installing an application through programming	324
Starting an application through programming	327
Uninstalling an application through programming	328
Manipulating additional attributes for a deployed application	331
Sharing sessions for application management	333
Updating an application through programming	334
Adding to, updating, or deleting part of an application through programming	336
Editing applications	338
Preparing a module and adding it to an existing application through programming	340
Preparing and updating a module through programming	343
Deleting a module through programming	346
Adding a file through programming	348
Updating a file through programming	350
Deleting a file through programming	353
Extending application management operations through programming	355
Chapter 9. Deploying and administering business-level applications	359
Business-level applications	359
Assets	362
Composition units	362
Importing assets	363
Upload asset settings	365
Asset settings	366
Managing assets	369
Asset collection.	369
Updating assets	370
Deleting assets	373
Exporting assets	373
Creating business-level applications	374
Creating business-level applications with the console	375
Business-level application settings.	384
Composition unit settings	386
Example: Creating a business-level application	387
Starting business-level applications	388
Stopping business-level applications	389
Updating business-level applications	389
Deleting business-level applications	391
Chapter 10. Administering business-level applications using programming	393
Creating an empty business-level application using programming	395
Importing an asset using programming	398
Adding a composition unit using programming	403
Starting a business-level application using programming.	410
Stopping a business-level application using programming	413
Checking the status of a business-level application using programming	416
Deleting a business-level application using programming	421
Deleting an asset using programming	424
Deleting a composition unit using programming	428
Exporting an asset using programming	432
Listing assets using programming	435
Listing composition units using programming	439

Listing business-level applications using programming	443
Editing a composition unit using programming	446
Editing an asset using programming	452
Editing a business-level application using programming	456
Updating an asset using programming	460
Viewing a composition unit using programming	464
Viewing an asset using programming.	468
Viewing a business-level application using programming.	471
Listing control operations using programming.	475
Chapter 11. Troubleshooting deployment	481
Application deployment problems	481
Application deployment troubleshooting tips	485
A client program does not work	486
Application startup errors	487
Application startup problems	491
Web resource is not displayed	494
Application uninstallation problems.	496
Chapter 12. Troubleshooting administration	497
Administration and administrative console troubleshooting	497
Administrative console does not start even though installation completes	501
Administrative console - browser connection problems	502
When a single user that uses multiple instances of the Mozilla browser logs into the administrative console, the first user ID that logs into the administrative console is the current user.	502
A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.	502
A user on Mozilla browser Version 1.4 enters an invalid ID or password, presses Enter, and receives an error message.	503
Web server plug-in troubleshooting tips	503
Administrative scripting problems	505
Tracing and logging facilities - troubleshooting tips	511
Server startup problems	512
Server hangs during shutdown if it creates a Java core dump (Red Hat Linux)	513
Command line tool problems	513
Appendix. Directory conventions	517
Notices	519
Trademarks and service marks	521

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
 1. Display the article in your Web browser and scroll to the end of the article.
 2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
 3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-5250.

Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Changes to serve you more quickly

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Under construction!

The Information Development Team for IBM WebSphere Application Server is changing its PDF book delivery strategy to respond better to user needs. The intention is to deliver the content to you in PDF format more frequently. During a temporary transition phase, you might experience broken links. During the transition phase, expect the following link behavior:

- Links to Web addresses beginning with `http://` work
- Links that refer to specific page numbers within the same PDF book work
- The remaining links will *not* work. You receive an error message when you click them

Thanks for your patience, in the short term, to facilitate the transition to more frequent PDF book updates.

Chapter 1. Overview and new features for administering applications and their environments

Use the links provided in this topic to learn about the administrative features.

What is new for administrators

This topic provides an overview of new and changed features of system administration.

“Introduction: System administration” on page 17

This topic describes the administration of the product and the applications that run on it.

See also “Introduction: Environment” on page 25 and “Introduction: Variables” on page 27.

Web applications

You create a Web module by assembling servlets, JavaServer Pages (JSP) files, and static content such as HyperText Markup Language (HTML) pages into a single deployable unit. You can then deploy the Web module as a stand-alone Web application, or combine it with other modules to create Java™ Platform, Enterprise Edition (Java EE) applications.

Whichever application type you deploy, you have the option of activating the **Session Manager** for your Web module. By using this service to manage HTTP sessions, you can personalize Web pages for individual users. A session is a series of requests to a servlet, originating from the same user at the same browser. Managing HTTP sessions allows servlets running in a Web container to keep track of individual users.

For example, a servlet might use sessions to provide “shopping carts” to on-line shoppers. Suppose the servlet is designed to record the items each shopper indicates he or she will purchase from the Web site. It is important that the servlet be able to associate incoming requests with particular shoppers. Otherwise, the servlet might mistakenly add choices of Shopper 1 to the cart of Shopper 2.

EJB applications

IBM WebSphere Application Server provides broad support for enterprise beans, including the Enterprise JavaBeans (EJB) 2.x and EJB 3.0 specifications. The EJB 2.x specification includes a container-managed persistence (CMP) 2.0 component model, which provides a number of improvements to aid developer productivity and application performance. The EJB 3.0 specification simplifies the development of business logic and data handling for enterprise applications.

Version support

The EJB 3.0 specification simplifies the development of business logic and data handling for enterprise applications. The EJB 2.x specification features a container-managed persistence (CMP) 2.0 component model, which provides a number of improvements to aid developer productivity and application performance.

In addition, this product continues to fully support enterprise beans written to the CMP 1.1 programming model and deployed in previous versions of this product; applications can use CMP 1.1 beans, CMP 2.0 beans, or a mixture of both. CMP 1.1 beans can be directly carried forward in an EJB 1.1 `ejb-jar` module or may be repackaged and combined with CMP 2.0 beans in an EJB 2.x module.

Several excellent trade books that cover the latest EJB specification and the CMP persistence model are available. A good way to locate some of these is to visit your favorite online bookstore and search on the term *Enterprise JavaBeans*.

Innovations for EJB 3.0 development

The purpose of the EJB 3.0 release is to make things simpler for the developer of enterprise applications. In particular, the new persistence model, called the Java Persistence API (JPA), is much simpler than the Container Managed Persistence (CMP) Entity bean approach from earlier releases. With a focus on the premise that a simple JavaBean can be used in most cases, and greater complexity should only be used when called for, EJB 3.0 simplifies the application development process in the following ways.

- Annotations to provide component metadata. It is no longer necessary to create Extensible Markup Language (XML) deployment descriptors.
- Programmatic defaults so you do not need to specify most commonly used requirements in the EJB container. You only need to specify an item if you do not want the default value.
- A dependency injection function and simpler lookup APIs for easier access to a bean's environment.
- Plain Old Java Object (POJO) development for both business logic and persistence. This enables unit test outside of the application server, reducing the edit-compile-deploy-debug cycles currently used with applications. It also removes dependencies on external framework interfaces such as *javax.ejb.SessionBean*, *javax.ejb.EJBObject*, and so on.
- No requirement for session beans to have *home* interfaces.
- Simplified entity persistence through the JPA. Light-weight domain modeling, including inheritance and polymorphism, is supported.

Refer to:

- “Data access resources” on page 6 An overview of data access.
- “Messaging resources” on page 7 An overview of messaging.

Innovations for EJB 2.x development

For EJB 2.x modules, a feature introduced in Version 5 of this product, called Access intent policies, eases the management of interactions between CMP beans and their underlying data stores. Each policy sets data access characteristics such as access type (read or update) and transaction isolation that affect the locking of resources, letting you choose the level of data integrity and performance for your application. The Integration Server product adds APIs to enable you to further customize IBM-provided access intent policies for your particular environment.

Access intent is frequently used with the function of application profiling. For example, you can configure one transaction to load an entity bean with strong update locks and configure another transaction to load the same entity bean without locks.

Sometimes when working with entity beans you might find that it is better to use the dynamic query service rather than the regular EJB query service (which can be referred to as *deployment query*).

Your application development can also include asynchronous messaging, which the product supports as a method of communication based on the Java Message Service (JMS) programming interface. The base JMS support enables product applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). An application can explicitly poll for messages on a destination. For more information, see Learning about messaging with WebSphere® Application Server.

The product also provides a message listener service that applications can use to automatically retrieve messages from JMS destinations for processing by message-driven beans, without the application having to explicitly poll JMS destinations.

Refer to:

- “Data access resources” on page 6 An overview of data access.
- “Messaging resources” on page 7 An overview of messaging.

Client modules

Application Clients for WebSphere Application Server follow several programming models with unique requirements and suitability for different types of applications. Most of these models are only available when you install the product.

Client modules

For a list of supported products and version information, see the Supported Prerequisites Web site.

J2EE application client

The J2EE application client programming model provides the benefits of Java 2 Platform, Enterprise Edition (J2EE).

Thin application client

This application client is a Java application client without the J2EE client container support. It uses a different programming model than the J2EE application client.

Related concepts

ActiveX application clients

WebSphere Application Server provides an ActiveX to EJB bridge that enables ActiveX programs to access enterprise beans through a set of ActiveX automation objects.

Applet clients

The applet client provides a browser-based Java run time capable of interacting with enterprise beans directly, instead of indirectly through a servlet.

Java EE application clients

The Java Platform, Enterprise Edition (Java EE) application client programming model provides the benefits of the Java EE Platform for WebSphere Application Server Enterprise product.

Pluggable application clients

The Pluggable application client provides a lightweight, downloadable Java application run time capable of interacting with enterprise beans.

Thin application clients

The Thin application client provides a lightweight, downloadable Java application run time capable of interacting with enterprise beans.

Web services

Web services are a technology that invoke services or applications using Internet standards and protocols. Implementing a service-oriented architecture, you can use Web services to perform encapsulated business functions ranging from simple request-reply to full business process interactions. Web services are self-contained, modular applications that you can describe, publish, locate, and invoke over a network. WebSphere Application Server implements a number of core technologies on which Web services are developed and implemented.

If you have an existing application, and you want to make the service that your application provides available to others - either within your own organization or beyond it - you can use Web services technologies to provide a standard Web interface for your service. Web services can be defined as middleware that connects applications together no matter how each application is implemented or where it is located.

Web services operate at a level of abstraction that is similar to the Internet; they can work with any operating system, hardware platform or programming language that can be Web-enabled and are one of the technologies that you can use to implement a service-oriented architecture (SOA).

The WebSphere® Application Server supports a set of Web services standards that support the creation and administration of interoperable, securable, transactionable, and reliable Web services applications.

Using the strategic Java™ API for XML-Based Web Services (JAX-WS) programming model, Web service clients can now additionally invoke Web services asynchronously, which means your client can continue processing without waiting on the response. Your JAX-WS Web services can also take advantage of the Web Services Reliable Messaging protocol quality of service where you can be confident that your communication is reliable and reaches its destination while interoperating with other vendors.

The core technologies on which Web services are developed and implemented include:

Programming models

Web Services for Java Platform, Enterprise Edition (Java EE)

This specification defines the programming model and runtime architecture for implementing Web services based on the Java language. WebSphere Application Server Version 7.0 supports Web Services for Java EE specification (JSR 109), Version 1.2.

If you want to know how to implement a Web service interface to an existing application, then deploy your Web service within the application server. To learn more, see *Implementing Web services applications*.

Java API for XML Web Services (JAX-WS)

This specification simplifies the development of Web services and clients with greater platform independence for Java applications by the use of dynamic proxies and Java annotations. WebSphere Application Server Version 7.0 supports JAX-WS 2.1.

Java API for XML-Based RPC (JAX-RPC)

JAX-RPC enables you to develop SOAP-based interoperable and portable Web services and Web services clients.

Java Architecture for XML Binding (JAXB)

This specification 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. WebSphere Application Server Version 7.0 supports JAXB 2.1.

SOAP with Attachments API for Java interface (SAAJ)

SAAJ is used for SOAP messaging that works behind the scenes in the Java API for XML Web Services (JAX-WS) or Java API for XML-based RPC (JAX-RPC) implementation. You can also use this API to directly write SOAP messaging applications rather than using JAX-WS or JAX-RPC. SAAJ enables you to do XML messaging from the Java platform by making method calls by creating, sending and consuming XML messages over the Internet.

A Web Services Invocation Framework (WSIF)

SOAP bindings for Web services are part of the WSDL specification. So when you think of using a Web service, you probably think of assembling a SOAP message and sending it across the network to the service endpoint, using some SOAP client API. The WSDL specification enables for extensibility points which can describe alternate ways of invoking a Web service. A WSIF client can make use of these non-SOAP descriptions to invoke a service in a more efficient way. For example, a Web service provider might offer a SOAP binding for the service and a local Java binding that enables you to treat the local service implementation (a Java class) as a Web service. If the client is deployed in the same environment as the service, then the local Java binding for the service can be used. This provides more efficient communication with the service by making direct Java calls rather than sending and receiving SOAP messages.

To deploy a Web service as a WSIF-enabled service, you first develop and deploy the Web service, then you develop the WSIF client based on the WSDL document for that Web service.

Web services standards

SOAP Message Transmission Optimization Mechanism (MTOM)

XML (Extensible Markup Language)

XML solves the problem of data independence. You use it to describe data, and also to map that data into and out of any application or programming language.

WSDL (Web Services Description Language)

You use this XML-based language to create a description of an underlying application. It is this description that turns an application into a Web service, by acting as the interface between the underlying application and other Web-enabled applications.

SOAP 1.1 and 1.2

SOAP is the core communications protocol for the Web, and most Web services use this protocol to talk to each other.

SOAP Qualities of Service

You can get different qualities of service with WS-ReliableMessaging, depending on the level of durability and transaction support provided by the store used to manage the reliable messaging state. These qualities of service range from protecting against loss of messages across a network, through to protecting against server failure.

WS-Security

IBM® supports Web services security, which is an extension of the IBM Web services engine, to provide a quality of service. The WebSphere Application Server security infrastructure fully integrates Web services security with the Java Platform, Enterprise Edition (Java EE) security specification.

WS-Transactions

WS-Transaction is an interoperability standard that includes the WS-AtomicTransaction, WS-BusinessActivity, and WS-Coordination specifications.

WS-ReliableMessaging

With WS-ReliableMessaging, you can make your SOAP over HTTP-based Web services interoperable and reliable without having to write custom code. For more information, see Adding assured delivery to Web services through WS-ReliableMessaging

WS-Policy

WS-Policy is an interoperability standard to describe and communicate the policies of a Web service so that service providers can export policy requirements in a standard format. Clients can combine the service provider requirements with their own capabilities to establish the policies required for a specific interaction.

WS-MetadataExchange

The WS-MetadataExchange specification defines a mechanism to retrieve metadata from an endpoint. WebSphere Application server supports the use of the WS-MetadataExchange 1.1 GetMetadata request to return metadata in a response.

WS-SecureExchange

WS-Secure Exchange is an effort to normalize the WS-SecureConversation, WS-SecurityPolicy and WS-Trust suite of specifications.

WS-Trust

Web Services Trust (WS-Trust) is a proposed Organization for the Advancement of Structured Information Standards (OASIS) standard that enables security token interoperability by defining a request/response protocol. This protocol enables SOAP actors, such as a Web services client, to request of some trusted authority that a particular security token be exchanged for another. The trust service, which is provided with WebSphere® Application Service, uses the secure messaging mechanisms of WS-Trust to define additional extensions for the issuance, exchange, and validation of security tokens.

Other Web services and related technologies

WS-Notification

WS-Notification enables Web services to use the “publish and subscribe” messaging pattern. In this pattern a producing application inserts (publishes) a message (event notification) into the messaging system, having marked it with a topic that indicates the subject area of the message. Consuming applications that have subscribed to the topic and have the appropriate authority, receive an independent copy of the message that was published by the producing application.

WS-Notification also enables interchange of event notification between WS-Notification applications and other clients of the service integration bus. By exploiting other service integration bus functionality, you can also use this function to interchange messages with other IBM publish and subscribe brokers such as Event Broker or Message Broker.

Service integration bus (SIBus) and platform messaging

Web services can use the SIBus to provide a single point of control, access, and validation of Web service requests and to enable control of Web services that are available to different groups of Web service users.

With SIBus-enabled Web services, you can achieve the following goals:

- Take an internally-hosted service that is available at a bus destination, and make it available as a Web service.
- Take an externally-hosted Web service, and make it available internally at a bus destination.

For more information, see *Enabling Web services through service integration technologies*

A private Universal Description, Discovery and Integration (UDDI) registry

A private UDDI registry provides a way to publish and discover information about Web services that are available within and through your organization. You can use it to make your Web services available to people within your organization, or beyond your organization. A group of companies can use it to share their Web services, or to make them available to others outside the group. At its simplest, a private UDDI registry does for Web services what a business telephone directory does for business addresses and telephone numbers. However, a private UDDI registry is much more than just a directory. It needs to be in order to harness the considerable power and flexibility of Web services. If you publish your Web service to UDDI, you make it available for other people or applications to discover and reuse. This saves development time, effort and cost, and helps minimize the need to maintain several different implementations of the same application.

Web services distributed management (WSDM)

WSDM is an OASIS approved standard that supports managing resources through a standardized Web service interface. Your environment, such as WebSphere® Application Server host or an operating system host that has an exposed resource as a Web service within a single interface is used to manage and control resources. WSDM is a distributed management model, but it does not replace any existing WebSphere Application Server administration models.

Streaming API for XML (StAX)

StAX is an efficient method to change and traverse XML data. In previous releases, if you wanted to manipulate XML schemas and types, you had to use one of two standard mechanisms for parsing your XML data: Documents Object Model (DOM) and Simple API for XML (SAX). StAX is another, more efficient, alternative to manipulate XML data.

Enterprise JavaBeans™ (EJB) 3.0

The EJB 3.0 specification provides the foundation of the development and application programming model for EJB 3.0 applications.

For a complete list of the supported standards and specifications, see the *Web services specifications and API documentation*.

Data access resources

These topics provide information about accessing data resources.

The connection between an enterprise application and an enterprise information system (EIS) is accomplished through the use of EIS-provided resource adapters, which are plugged into the application server. The resource adapter plays a central role in the integration and connectivity between an EIS and an application server. It serves as the point of contact between application components, application servers, and enterprise information systems. A resource adapter must communicate with other components based on well-defined contracts that are specified by the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA).

Consult the following concept, reference, and task files for more overview information.

Messaging resources

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) and Java EE Connector Architecture (JCA) programming interfaces. WebSphere Application Server also supports the use of message-driven beans as asynchronous message consumers.

The JMS and JCA programming interfaces provide a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests as messages. The application explicitly polls the destination for messages whereas a message-driven bean is invoked by the EJB container when a message arrives at a pre-configured destination.

To handle non-JMS requests inbound to WebSphere Application Server from enterprise information systems, message-driven beans use a JCA 1.5 resource adapter written for that purpose. In the JCA 1.5 specification, such message-driven beans are commonly called *message endpoints* or simply *endpoints*. You use a J2C activation specification to configure such message-driven beans as JCA 1.5 resources.

Message-driven beans that implement the `javax.jms.MessageListener` interface can be used for JMS messaging. For JMS messaging, message-driven beans can use a JCA-based messaging provider such as the default messaging provider that is part of WebSphere Application Server. You use a J2C activation specification to configure such message-driven beans as JCA 1.5 resources. For compatibility with WebSphere Application Server Version 5, you can configure JMS message-driven beans against a listener port.

You can use the WebSphere Application Server administrative console to administer the WebSphere Application Server support for asynchronous messaging. For example, you can configure JCA resource, adapters, J2C activation specifications, JMS providers, and JMS resources, and can control the activity of messaging services.

Tools for working with messaging include the following:

- Rational® Application Developer for developing and packaging J2EE applications that use JMS, message-driven beans, and other programming techniques for messaging.
- Assembly tools for working with the packaging and deployment descriptors for WebSphere Application Server applications, as described in *Assembling applications*
- Product system administration tools for installing and managing WebSphere Application Server applications, as described in “Introduction: System administration” on page 17

For more information about implementing enterprise applications that use asynchronous messaging, see the following topics:

- Introduction: Messaging resources
- Choosing a messaging provider
- Managing messaging with the default messaging provider
- Managing messaging with WebSphere MQ as your JMS provider
- Managing messaging with a third-party messaging provider
- Administering listener ports and activation specifications for message-driven beans
- Programming to use asynchronous messaging

- Troubleshooting WebSphere messaging

Mail, URLs, and other J2EE resources

The product supports all of the resources defined by the Java Platform, Enterprise Edition (Java EE). It adds the following resources in support of service extensions:

- Schedulers
- Work managers
- Object pools

Data access (JDBC and J2C)

The J2EE Connector architecture defines a standard architecture that enables the integration of various enterprise information systems (EIS) with application servers and enterprise applications. It defines a standard resource adapter used by a Java application to connect to an EIS. This resource adapter can plug into the application server and, through the Common Client Interface (CCI), provide connectivity between the EIS, the application server, and the enterprise application.

For more information, refer to “Data access resources” on page 6.

Messaging

The product supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. The base JMS support enables the product applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics).

For more information, refer to “Messaging resources” on page 7.

Mail

Using JavaMail API, a code segment can be embedded in any Java EE application component, such as an EJB or a servlet, allowing the application to send a message and save a copy of the mail to the Sent folder.

For more information, refer to JavaMail API.

URLs

Java EE applications can use URLs as resources in the same way other Java EE resources, such as JDBC and JavaMail, are used.

For more information, refer to URLs.

Resource environment entries

A resource environment reference maps a logical name used by the client application to the physical name of an object.

For more information, see Mapping logical names of environment resources to their physical names.

Security

The following information provides an overview of security in WebSphere Application Server.

IBM WebSphere Application Server provides security infrastructure and mechanisms to protect sensitive Java 2 Platform, Enterprise Edition (J2EE) resources and administrative resources. It also addresses enterprise end-to-end security requirements on:

- Authentication
- Resource access control
- Data integrity
- Confidentiality
- Privacy
- Secure interoperability

IBM WebSphere Application Server security is based on industry standards and has an open architecture that ensures secure connectivity and interoperability with Enterprise Information Systems (EIS) including:

- Database 2™ (DB2®)
- CICS®
- MQ Series
- Lotus® Domino®
- IBM Directory

WebSphere Application Server also supports other security providers including:

- IBM Tivoli® Access Manager (Policy Director)
- Reverse secure proxy server including WebSEAL

Based on industry standards

IBM WebSphere Application Server provides a unified, policy-based, and permission-based model for securing Web resources, Web service endpoints, and enterprise JavaBeans according to J2EE specifications. Specifically, WebSphere Application Server complies with J2EE specification Version 1.4 and has passed the J2EE Compatibility Test Suite.

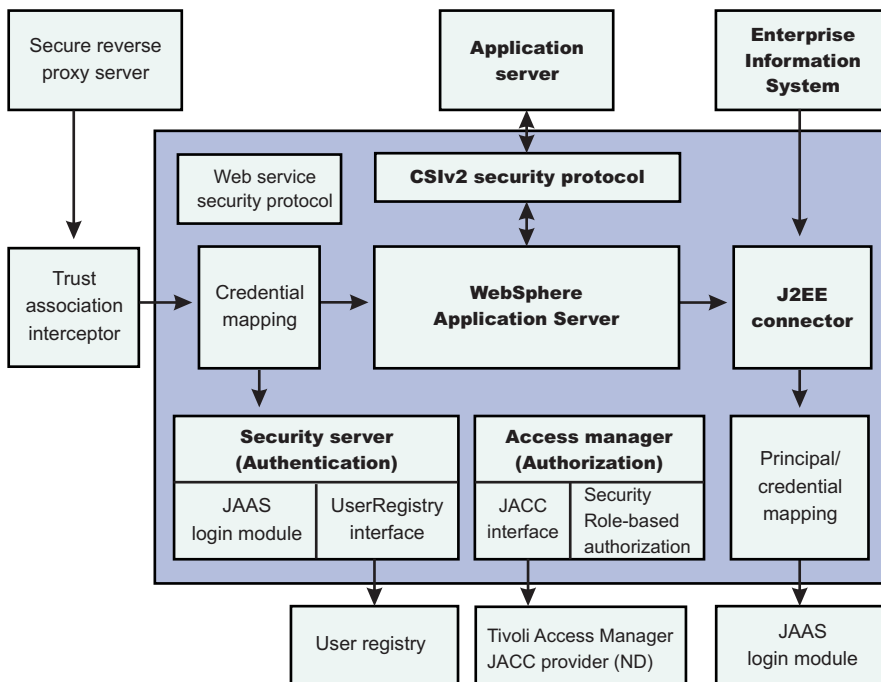
WebSphere Application Server security is a layered architecture built on top of an operating system platform, a Java virtual machine (JVM), and Java 2 security. This security model employs a rich set of security technology including the:

- Java 2 security model, which provides policy-based, fine-grained, and permission-based access control to system resources.
- Common Secure Interoperability Version 2 (CSIv2) security protocol, in addition to the Secure Authentication Services (SAS) security protocol. Both protocols are supported by prior WebSphere Application Server releases. CSIv2 is an integral part of the J2EE 1.4 specification and is essential for interoperability among application servers from different vendors and with enterprise CORBA services.
- Java Authentication and Authorization Service (JAAS) programming model for Java applications, servlets, and enterprise beans.
- J2EE Connector architecture for plugging in resource adapters that support access to Enterprise Information Systems.

The standard security models and interfaces that support secure socket communication, message encryption, and data encryption are the Java Secure Socket Extension (JSSE) and the Java Cryptographic Extension (JCE).

Open architecture paradigm

An application server plays an integral part in the multiple-tier enterprise computing framework. IBM WebSphere Application Server adopts the open architecture paradigm and provides many plug-in points to integrate with enterprise software components. Plug-in points are based on standard J2EE specifications wherever applicable.



The dark blue shaded background indicates the boundary between the WebSphere Application Server Version 7.0 and other business application components.

WebSphere Application Server provides Simple WebSphere Authentication Mechanism (SWAM), Lightweight Third Party Authentication (LTPA), and Kerberos as the authentication mechanism. Exactly one authentication mechanism can be configured to be the active authentication mechanism for the security domain of WebSphere Application Server. Exactly one user registry implementation can be configured to be the active user registry of WebSphere Application Server security domain. WebSphere Application Server provides the following user registry implementations: UNIX®, Windows®, and i5/OS® local OS and Lightweight Directory Access Protocol (LDAP). It also provides file-based and Java database connectivity (JDBC)-based user registry reference implementations. It supports a flexible combination of authentication mechanisms and user registries. SWAM is simple to configure and is useful for a single application server environment. LTPA generates a security token for authenticated users. It is possible to use SWAM in a distributed environment if identity assertion is enabled. The identity assertion feature is available only on the CSiv2 security protocol.

Note: SWAM was deprecated in WebSphere Application Server Version 7.0 and will be removed in a future release.

The LTPA authentication mechanism is designed for distributed security. Downstream servers can validate the security token. It also supports setting up a trust association relationship with reverse secure proxy servers and single sign-on (SSO), which is discussed later. Besides the combination of LTPA and LDAP or Custom user registry interface, Version 6.x or higher supports LTPA with a LocalOS user registry interface. The new configuration is particularly useful for a single node with multiple application servers. It can function in a distributed environment if the local OS user registry implementation is a centralized user registry, such as Windows Domain Controller, or can be maintained in a consistent state on multiple nodes.

WebSphere Application Server supports the J2EE Connector architecture and offers container-managed authentication. It provides a default Java 2 Connector (J2C) principal and credential mapping module that maps any authenticated user credential to a password credential for the specified Enterprise Information Systems (EIS) security domain. The mapping module is a special JAAS login module designed according to the Java 2 Connector and JAAS specifications. Other mapping login modules can be plugged in.

Web services security

WebSphere Application Server enables you to secure Web services based upon the Organization for the Advancement of Structured Information Standards (OASIS) Web services security Version 1.1 specification. These standards address how to provide protection for messages exchanged in a Web service environment. The 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.

Trust associations

Trust association enables you to integrate IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the WebSphere Application Server applies its own authorization policy onto the resulting credentials that are passed by the proxy server. The reverse proxy server applies its authentication policies to every Web request that is dispatched to WebSphere Application Server. The products that implement trust association interceptors (TAI) include:

- IBM Tivoli Access Manager for e-business
- WebSEAL
- Caching Proxy

For more information on using trust association, refer to Trust associations.

Security attribute propagation

Security attribute propagation enables WebSphere Application Server to transport security attributes from one server to another in your configuration. Security attributes include authenticated subject contents and security context information. WebSphere Application Server can obtain these security attributes from either:

- An enterprise user registry that queries static attributes
- A custom login module that can query static or dynamic attributes

Security attribute propagation provides propagation services using Java serialization for any objects that are contained in the subject. For more information on using security attribute propagation, refer to Security attribute propagation.

Single sign-on interoperability mode

In WebSphere Application Server, the interoperability mode option enables Single Sign-on (SSO) connections between WebSphere Application Server version 5.1.1 or later to interoperate with previous versions of the application server. When you select this option, WebSphere Application Server adds the old-style LtpaToken into the response so that it can be sent to other servers that work only with this token type. This option applies only when the Web inbound security attribute propagation option is enabled. For more information on single sign-on, refer to Implementing single sign-on to minimize Web user authentications

Security for J2EE resources using Web containers and EJB containers

Each container provides two kinds of security: *declarative security* and *programmatic security*. In declarative security, the security structure of an application, including data integrity and confidentiality, authentication requirements, security roles, and access control, is expressed in a form external to the application. In particular the deployment descriptor is the primary vehicle for declarative security in the J2EE platform. WebSphere Application Server maintains a J2EE security policy, including information derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At run time, the container uses the security policy defined in the XML descriptor files to enforce data constraints and access control. When declarative security alone is not sufficient to express

the security model of an application, the application code can use programmatic security to make access decisions. The application programming interface (API) for programmatic security consists of two methods of the Enterprise JavaBeans (EJB) EJBContext interface (`isCallerInRole`, `getCallerPrincipal`) and three methods of the servlet `HttpServletRequest` interface (`isUserInRole`, `getUserPrincipal`, `getRemoteUser`).

Web security

When a security policy is specified for a Web resource and IBM WebSphere Application Server security is enforced, the Web container performs access control when the resource is requested by a Web client. The Web container challenges the Web client for authentication data if none is present according to the specified authentication method, ensures that the data constraints are met, and determines whether the authenticated user has the required security role. WebSphere Application Server supports the following login methods:

- HTTP basic authentication
- HTTPS client authentication
- Form-based Login

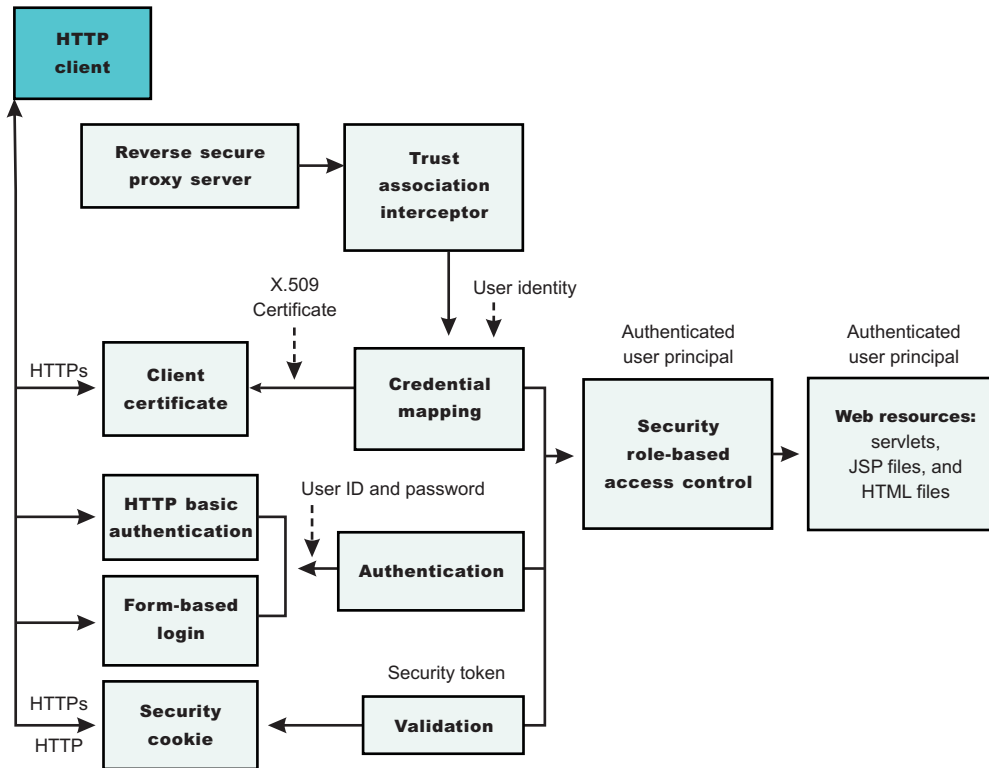
Mapping a client certificate to a WebSphere Application Server security credential uses the `UserRegistry` implementation to perform the mapping.

On WebSphere Application Server Express, the local OS user registry does not support the mapping function.

When the LTPA authentication mechanism is configured and single sign-on (SSO) is enabled, an authenticated client is issued a security cookie, which can represent the user within the specified security domain.

It is recommended that you use Secure Sockets Layer (SSL) to protect the security cookie or Basic Authentication information from being intercepted and replayed. When a trust association is configured, WebSphere Application Server can map an authenticated user identity to security credentials based on the trust relationship established with the secure reverse proxy server.

Web Security



When considering Web security collaborators and EJB security collaborators:

1. The Web security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions based on the security policy derived from the deployment descriptor. An authenticated user principal can access the requested Servlet or JSP file if the user principal has one of the required security roles. Servlets and JSP files can use the `HttpServletRequest` methods: `isUserInRole`, `getUserPrincipal`, and `getRemoteUser`. As an example, the administrative console uses the `isUserInRole` method to determine the proper set of administrative functionality to expose to a user principal.
2. The EJB security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions based on the security policy derived from the deployment descriptor. An authenticated user principal can access the requested EJB method if it has one of the required security roles. EJB code can use the `EJBContext` methods `isCallerInRole` and `getCallerPrincipal`. EJB code also can use the JAAS programming model to perform JAAS login and `WSSubject doAs` and `doAsPrivileged` methods. The code in the `doAs` and `doAsPrivileged PrivilegedAction` block executes under the `Subject` identity. Otherwise, the EJB method executes under either the `RunAs` identity or the caller identity, depending on the `RunAs` configuration.

EJB security

When security is enabled, the EJB container enforces access control on EJB method invocation. The authentication takes place regardless of whether a method permission is defined for the specific EJB method.

A Java application client can provide the authentication data in several ways. Using the `sas.client.props` file, a Java client can specify whether to use a user ID and password to authenticate or to use an SSL client certificate to authenticate. The client certificate is stored in the key file or in the hardware cryptographic card, as defined in a `sas.client.props` file. The user ID and password can be optionally defined in the `sas.client.props` file.

At run time, the Java client can either perform a programmatic login or perform a *lazy authentication*.

In lazy authentication when the Java client is accessing a protected enterprise bean for the first time, the security runtime tries to obtain the required authentication data. Depending on the configuration setting in `sas.client.props` file the security runtime either looks up the authentication data from this file or prompts the user. Alternatively, a Java client can use programmatic login. WebSphere Application Server supports the JAAS programming model and the JAAS login (`LoginContext`) is the recommended way of programmatic login. The `login_helper request_login` helper function is deprecated in Version 6.x and Version 7.0. Java clients programmed to the `login_helper` APT can run in this version.

The EJB security collaborator enforces role-based access control by using an access manager implementation.

An access manager makes authorization decisions based on the security policy derived from the deployment descriptor. An authenticated user principal can access the requested EJB method if it has one of the required security roles. EJB code can use the `EJBContext` methods `isCallerInRole` and `getCallerPrincipal`. EJB code also can use the JAAS programming model to perform JAAS login and `WSSubject doAs` and `doAsPrivileged` methods. The code in the `doAs` and `doAsPrivileged` `PrivilegedAction` block executes under the Subject identity. Otherwise, the EJB method executes under either the `RunAs` identity or the caller identity, depending on the `RunAs` configuration. The J2EE `RunAs` specification is at the enterprise bean level. When `RunAs` identity is specified, it applies to all bean methods. The method level IBM `RunAs` extension introduced in Version 4.0 is still supported in this version.

Federal Information Processing Standards-approved

Federal Information Processing Standards (FIPS) are standards and guidelines issued by the National Institute of Standards and Technology (NIST) for federal computer systems. FIPS are developed when there are compelling federal government requirements for standards, such as for security and interoperability, but acceptable industry standards or solutions do not exist.

WebSphere Application Server integrates cryptographic modules including Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE), which have undergone FIPS 140-2 certification. Throughout the documentation and the WebSphere Application Server, the IBM JSSE and JCE modules that have undergone FIPS certification are referred to as `IBMJSSEFIPS` and `IBMJCEFIPS`, which distinguishes the FIPS modules from the IBM JSSE and IBM JCE modules.

For more information, refer to `Configuring Federal Information Processing Standard Java Secure Socket Extension` files.

The `IBMJSSEFIPS` module supports the FIPS-approved Transport Layer Security (TLS) cipher suites including:

- SHA
- DES
- TripleDES

The `IBMJSSEFIPS` module supports the following algorithms:

- RSA public key algorithm
- ANSI X9.31
- IBM Random Number Generator (Patent pending)

The `IBMJCEFIPS` module supports the following symmetric cipher suites:

- AES (FIPS 197)
- TripleDES (FIPS 46-3)
- SHA1 Message Digest algorithm (FIPS 180-1)

The IBMJCEFIPS module supports the following algorithms:

- Digital Signature DSA and RSA algorithms (FIPS 186-2)
- ANSI X 9.31 (FIPS 186-2)
- IBM Random Number Generator

The IBMJCEFIPS cryptographic module contains the algorithms that are approved by FIPS, which form a proper subset of those in the IBM JCE modules.

Related concepts

Access control exception

The Java 2 security behavior is specified by its *security policy*. The security policy is an access-control matrix that specifies which system resources certain code bases can access and who must sign them. The Java 2 security policy is declarative and it is enforced by the `java.security.AccessController.checkPermission` method.

Common Secure Interoperability Version 2 features

The following Common Secure Interoperability Version 2 (CSIv2) features are available in IBM WebSphere Application Server: message layer authentication, identity assertion, and security attribute propagation.

Delegations

Delegation is a process security identity propagation from a caller to a called object. As per the Java Platform, Enterprise Edition (Java EE) specification, a servlet and enterprise beans can propagate either the client or remote user identity when invoking enterprise beans, or they can use another specified identity as indicated in the corresponding deployment descriptor.

Administrative security

Administrative security determines whether security is used at all, the type of registry against which authentication takes place, and other values, many of which act as defaults. Proper planning is required because incorrectly enabling administrative security can lock you out of the administrative console or cause the server to end abnormally.

Java Authentication and Authorization Service

The standard Java 2 security application programming interface (API) helps enforce access control based on the location of the code source or the author or packager of the code that signed the jar file. The current principal of the running thread is not considered in the Java 2 security authorization. Instances where authorization is based on the principal, as opposed to the code base, and the user exist. The Java Authentication and Authorization Service is a standard Java API that supports the Java 2 security authorization to extend the code base on the principal as well as the code base and users.

J2EE connector security

The Java 2 Platform, Enterprise Edition (J2EE) connector architecture defines a standard architecture for connecting J2EE to heterogeneous enterprise information systems (EIS). Examples of EIS include Enterprise Resource Planning (ERP), mainframe transaction processing (TP) and database systems.

Standalone Lightweight Directory Access Protocol registries

A Standalone Lightweight Directory Access Protocol (LDAP) registry performs authentication using an LDAP binding.

Local operating system registries

With the registry implementation for the local operating system, the WebSphere Application Server authentication mechanism can use the user accounts database of the local operating system.

Lightweight Third Party Authentication

Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. LTPA supports forwardable credentials and single sign-on (SSO). LTPA can support security in a distributed environment through cryptography. This support permits LTPA to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature.

Programmatic login

Programmatic login is a type of form login that supports application presentation site-specific login forms for the purpose of authentication.

Role-based authorization

Use authorization information to determine whether a caller has the necessary privileges to request a service.

Java 2 security

Java 2 security provides a policy-based, fine-grain access control mechanism that increases overall system integrity by checking for permissions before allowing access to certain protected system resources. Java 2 security guards access to system resources such as file I/O, sockets, and properties. Java 2 Platform, Enterprise Edition (J2EE) security guards access to Web resources such as servlets, JavaServer Pages (JSP) files and Enterprise JavaBeans (EJB) methods.

Trust associations

Trust association enables the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials that are passed by the proxy server.

Related tasks

Selecting an authentication mechanism

An *authentication mechanism* defines rules about security information, such as whether a credential is forwardable to another Java process, and the format of how security information is stored in both credentials and tokens. You can select and configure an authentication mechanism by using the administrative console.

Selecting a registry or repository

Information about users and groups reside in a user registry. In WebSphere Application Server, a user registry authenticates a user and retrieves information about users and groups to perform security-related functions, including authentication and authorization.

Configuring Federal Information Processing Standard Java Secure Socket Extension files

Use this topic to configure Federal Information Processing Standard Java Secure Socket Extension files.

Securing Web services applications using message level security

Web services security standards and profiles describe how to provide security and protection for SOAP messages that are exchanged in a Web services environment.

Securing Web services for Version 5.x applications based on WS-Security

Web services security for WebSphere Application Server is based on standards included in the Web Services Security (WS-Security) specification. These standards address how to provide protection for messages exchanged in a Web service environment.

Related reference

Java 2 security policy files


The Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 and later specifications have a well-defined programming model of responsibilities between the container providers and the application code. Using Java 2 security manager to help enforce this programming model is recommended. Certain operations are not supported in the application code because such operations interfere with the behavior and operation of the containers. The Java 2 security manager is used in the product to enforce responsibilities of the container and the application code.

Related information

Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List

Introduction: System administration

You can administer your WebSphere Application Server through scripts, command line tools, the administrative console, or the Java programming interface. You administer server processes, topological units referenced as nodes and cells, and the configuration repository where configuration information is stored in Extensible Markup Language (XML) files.

Note:  If you would prefer to browse PDF versions of this documentation using your Adobe® Reader, see the **System Administration** PDF files available from www.ibm.com/software/webservers/appserv/infocenter.html.

A variety of tools, processes, and configuration files are provided for administering the WebSphere Application Server product:

- **Console**

The administrative console is a graphical interface that provides many features to guide you through deployment and systems administration tasks. Use it to explore available management options.

For more information, refer to “Introduction: Administrative console” on page 18.

- **Scripting**

The WebSphere administrative (wsadmin) scripting program is a powerful, non-graphical command interpreter environment enabling you to run administrative operations in a scripting language. You can also submit scripting language programs to run in batch mode. The wsadmin tool is intended for production environments and unattended operations.

For more information, refer to “Introduction: Administrative scripting (wsadmin)” on page 19.

- **Command line tools**

Command-line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks, as opposed to general purpose administration. Using the tools, you can start and stop application servers, check server status, add or remove nodes, and complete similar tasks.

For more information, refer to “Introduction: Administrative commands” on page 19.

- **Programming**

The product supports a Java programming interface for developing administrative programs. All of the administrative tools supplied with the product are written according to the API, which is based on the industry standard Java Management Extensions (JMX) specification.

For more information, refer to “Introduction: Administrative programs” on page 20.

- **Data**

Product configuration data resides in XML files that are manipulated by the previously-mentioned administrative tools.

For more information, refer to “Introduction: Administrative configuration data” on page 20.

- **Product library, directories, subsystem, job queue, job description, and output queues**

An i5/OS platform uses different configurations than installations of WebSphere Application Server on other platforms.

For more information, refer to “Product library, directories, subsystem, job queue, job description, and output queues” on page 20.

Introduction: Administrative console

The administrative console is a graphical interface that allows you to manage your applications and perform system administration tasks for your WebSphere Application Server environment. The administrative console runs in your Web browser.

Your actions in the console modify a set of XML configuration files.

You can use the administrative console to perform tasks such as:

- Add, delete, start, and stop application servers
- Deploy new applications to a server
- Start and stop existing applications, and modify certain configurations
- Add and delete Java Platform, Enterprise Edition (Java EE) resource providers for applications that require data access, mail, URLs, and so on
- Configure product security, including access to the administrative console
- Collect data for performance and troubleshooting purposes
- Find the product version information. It is located on the front page of the console.

“Starting and logging off the administrative console” on page 32 helps you begin using the console so that you can explore the available options. See also the **Reference > Administrator > Settings** section of the information center navigation. It lists the settings or properties you can configure.

Introduction: Administrative scripting (wsadmin)

About this task

The WebSphere administrative (wsadmin) scripting program is a powerful, non-graphical command interpreter environment enabling you to run administrative operations in a scripting language. The wsadmin tool is intended for production environments and unattended operations. You can use the wsadmin tool to perform the same tasks that you can perform using the administrative console.

The following list highlights the topics and tasks available with scripting:

- Getting started with scripting Provides an introduction to WebSphere Application Server scripting and information about using the wsadmin tool. Topics include information about the scripting languages and the scripting objects, and instructions for starting the wsadmin tool.
- Using the Jython script library The script library provides Jython script procedures to assist in automating your environment. Use the sample scripts to manage applications, resources, servers, nodes, and clusters. You can also use the script procedures as examples to learn the Jython syntax.
- Deploying applications Provides instructions for deploying and uninstalling applications. For example, stand-alone Java archive files and Web archive files, the administrative console, remote enterprise archive (EAR) files, file transfer applications, and so on.
- Managing deployed applications Includes tasks that you perform after the application is deployed. For example, starting and stopping applications, checking status, modifying listener address ports, querying application state, configuring a shared library, and so on.
- Configuring servers Provides instructions for configuring servers, such as creating a server, modifying and restarting the server, configuring the Java virtual machine, disabling a component, disabling a service, and so on.
- Configuring connections to Web servers Includes topics such as regenerating the plug-in, creating new virtual host templates, modifying virtual hosts, and so on.
- Managing servers Includes tasks that you use to manage servers. For example, stopping nodes, starting and stopping servers, querying a server state, starting a listener port, and so on.
- Configuring security Includes security tasks such as enabling and disabling security.
- Configuring data access Includes topics such as configuring a Java DataBase Connectivity (JDBC) provider, defining a data source, configuring connection pools, and so on.
- Configuring messaging Includes topics about messaging, such as Java Message Service (JMS) connection, JMS provider, WebSphere queue connection factory, MQ topics, and so on.
- Configuring mail, URLs, and resource environment entries Includes topics such as mail providers, mail sessions, protocols, resource environment providers, referenceables, URL providers, URLs, and so on.
- Dynamic caching Includes caching topics, for example, creating, viewing and modifying a cache instance.
- Troubleshooting Provides information about how to troubleshoot using scripting. For example, tracing, thread dumps, profiles, and so on.
- Obtaining product information Includes tasks such as querying the product identification.
- Scripting reference material Includes all of the reference material related to scripting. Topics include the syntax for the wsadmin tool and for the administrative command framework, explanations and examples for all of the scripting object commands, the scripting properties, and so on.

Introduction: Administrative commands

Command line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks, as opposed to general purpose administration. Using the tools, you can start and stop application servers, check server status, add or remove nodes, and complete similar tasks.

See **Reference > Administrator > Commands** in the information center navigation area for the names and syntax of all the commands that are available with the product. A subset of these commands is particular to system administration purposes.

Introduction: Administrative programs

The Java Management Extensions (JMX) specification allows you to write Java programs to administer WebSphere Application Server.

The product supports a Java programming interface for developing administrative programs. All of the administrative tools supplied with the product are written according to the API, which is based on the industry standard Java Management Extensions (JMX) specification. You can write a Java program that performs any of the administrative features of the WebSphere Application Server administrative tools. You can also extend the basic WebSphere Application Server administrative system to include your own managed resources.

Introduction: Administrative configuration data

WebSphere Application Server configuration data is kept in files. All administrative actions that you perform involve changes to these files.

Administrative tasks typically involve defining new configurations of the product or performing operations on managed resources within the environment. WebSphere Application Server configuration data is kept in files. Because all product configuration involves changing the content of those files, it is useful to know the structure and content of the configuration files.

The WebSphere Application Server product includes an implementation of the Java Management Extension (JMX) specification. All operations on managed resources in the product go through JMX functions. This setup means a more standard framework underlying your administrative operations as well as the ability to tap into the systems management infrastructure programmatically.

Product library, directories, subsystem, job queue, job description, and output queues

An i5/OS platform uses different configurations than installations of WebSphere Application Server on other platforms.

This topic describes the product library, directories, subsystem, job queue, job description, and output queues that WebSphere Application Server uses on the i5/OS platform.

Product library and directories

In a default installation, all editions of WebSphere Application Server for i5/OS use the following library and directories:

QWAS7 library

Product library

/QIBM/UserData/WebSphere/AppServer/V7 directory

Root directory for user data that is common to all editions of the product.

/QIBM/ProdData/WebSphere/AppServer/V7/Base directory

Root directory containing product data that is shared by all WebSphere Application Server profiles. Additionally, the master copies of the files, which make up a profile, are stored here. Do not modify files under this directory structure.

/QIBM/UserData/WebSphere/AppServer/V7/Base directory

Root directory for profile information. If the `-profilePath` option is not specified when creating a profile, the profile is created under this path.

Subsystem

All editions of the product for i5/OS use the QWAS7 subsystem. The subsystem description for QWAS7 is located in the QWAS7 product library. The application server, node agent and deployment manager processes run in this subsystem.

Job queue

All editions of the product for i5/OS use the QWASJOBQ job queue. The job queue is in the QWAS7 product library. The application server, node agent, and deployment manager processes are all submitted to this job queue when you invoke the startServer, startNode, or startManager scripts. The QWASJOBQ job queue is associated with the QWAS7 subsystem.

Job description

All editions of the product for i5/OS use the QWASJOBQ job description. The job description is in the QWAS7 product library. The application server, node agent, and deployment manager processes use this job description when submitted to run in the QWAS7 subsystem.

The QWASJOBQ job description has the following default settings:

```
                Display Job Description
Job description:  QWASJOBQ      Library:  QWAS7      System:  MYISERIES

User profile . . . . . : QEJBSVR
CL syntax check . . . . . : *NOCHK
Hold on job queue . . . . . : *NO
End severity . . . . . : 30
Job date . . . . . : *SYSVAL
Job switches . . . . . : 00000000
Inquiry message reply . . . . . : *RQD
Job priority (on job queue) . . . . . : 5
Job queue . . . . . : QWASJOBQ
  Library . . . . . : QWAS7
Output priority (on output queue) . . . . . : 5
Printer device . . . . . : *USRPRF
Output queue . . . . . : QWASOUTQ
  Library . . . . . : QWAS7
Message logging:
  Level . . . . . : 4
  Severity . . . . . : 0
  Text . . . . . : *SECLVL
Log CL program commands . . . . . : *NO
Accounting code . . . . . : *USRPRF
Print text . . . . . : *SYSVAL

Routing data . . . . . : QCMDI

Request data . . . . . : *NONE

Device recovery action . . . . . : *SYSVAL
Time slice end pool . . . . . : *SYSVAL
Job message queue maximum size . . . . . : *SYSVAL
Job message queue full action . . . . . : *WRAP
Allow multiple threads . . . . . : *YES
Initial ASP group . . . . . : *NONE
Spooled file action . . . . . : *SYSVAL
Text . . . . . : WEBSPPHRE JOB DESCRIPTION
```

Use the **Request data** parameter to specify information to start a WebSphere Application Server profile when the QWAS7 subsystem is started. For more information on how to start an application server when the subsystem starts see Starting the product environment. You can modify the job description if the

values do not meet your requirements. However, you must not change the **Allow multiple threads** value. Because the WebSphere Application Server processes contain a Java virtual machine that is multithreaded, this value must be *YES.

Output queues

All editions of the product for i5/OS use the QWASOUTQ output queue. The output queue is in the QWAS7 product library. By default, this output queue stores job logs for these processes:

- The application server process
- The node agent process
- The deployment manager process

This queue also stores other spooled files. The public authority to the output queue is *USE. This setting ensures that the user profile that is specified for the USER parameter (on the SBMJOB command) has the necessary authority to the output queue when it submits the WebSphere Application Server job to the job queue. The default user profile is QEJBSVR.

You might want to clear the output queue periodically if the number or size of the spooled files becomes large.

Introduction: Application servers

Application servers provide the core functionality of the WebSphere Application Server product family. Application servers extend the ability of a Web server to handle Web application requests, and much more. An application server enables a server to generate a dynamic, customized response to a client request.

Introduction: Application servers

An application server is a Java Virtual Machine (JVM) that runs user applications. The application server collaborates with the Web server to return a dynamic, customized response to a client request. The client request can consist of servlets, JavaServer Pages (JSP) files, and enterprise beans, and their supporting classes.

For example, a user at a Web browser visits a company Web site:

1. The user requests access to data in a database.
2. The user request flows to the Web server.
3. The Web server determines that the request involves an application containing resources not handled directly by the Web server (such as servlets). It forwards the request to one of its application servers on which the application is running.
4. The invoked application then processes the user request. For example:
 - An application servlet prepares the user request for processing by an enterprise bean that performs the database access.
 - The application produces a dynamic Web page containing the results of the user query.
5. The application server collaborates with the Web server to return the results to the user at the Web browser.

When you install the product, a default application server, named server1, is automatically created. You can use the administrative console to manage this server.

You can use the administrative console or command-line tools to create additional application servers that can either be separately configured processes or nearly identical clones. You must either use command-line tools to manage these additional servers, set up an administrative console for each server, or configure an administrative agent to provide a single interface to all of your servers, including the original base server. An administrative agent makes it easier to more fully administer these unfederated application servers.

Note: If you create additional application servers, only use one server to modify and save configurations. There is no coordination of configuration setting between the different servers and if you modify and save configurations on multiple servers, your data might become corrupted.

Note: You can improve system performance if you configure some of your application servers, such that each of their components are dynamically started as they are needed, instead of letting all of these components automatically start when the server starts. Selecting this option can improve server startup time, and reduce the memory footprint. Starting components as they are needed is most effective if all of the applications that are deployed on the application server are of the same type. For example, using this option works better if all of your applications are Web applications that use servlets, and JavaServer Pages (JSP). This option works less effectively if your applications use servlets, JSPs and Enterprise JavaBeans (EJB).

You can also perform the following tasks to enhance the operation of an application server:

- Configure transport chains to provide network protocol support for I/O operations within an application server environment.
- Add an interface to an application server to define a hook point that runs when the server starts and shuts down.
- Define command-line information that passes to a server when it starts or initializes.
- Tune the application server.
- Enhance the performance of the application server JVM.
- Configure an Object Request Broker (ORB) for RMI/IIOP communication.

Asynchronous messaging

The product supports asynchronous messaging based on the Java Message Service (JMS) of a JMS provider that conforms to the JMS specification Version 1.1.

The JMS functions of the default message service that is provided with the product are served by one or more messaging engines (in a service integration bus) that runs within application servers.

Generic Servers

A generic server is a server that is managed in the WebSphere administrative domain, although it is not a server that is supplied by the product. The generic server can be any server or process that is necessary to support the product environment.

Related tasks

Creating generic servers

A generic server is a server that is managed in the z/OS® Application Server administrative domain even though the server is not a server that is supplied by WebSphere Application Server. The WebSphere Application Server generic servers function enables you to define a generic server as an application server instance within the WebSphere Application Server administration, and associate it with a non-WebSphere WebSphere Application server or process.

Programming to use JMS and messaging directly

Use these tasks to implement WebSphere J2EE applications that use JMS programming interfaces directly.

Administering application servers

An application server configuration provides settings that control how an application server provides services for running applications and their components.

Configuring transport chains

A transport chain consists of one or more types of channels, each of which supports a different type of I/O protocol, such as TCP or HTTP. Network ports can be shared among all of the channels within a chain. The channel framework function automatically distributes a request arriving on that port to the correct I/O protocol channel for processing.

Creating custom services

You can create one or more custom services for an application server. Each custom services defines a class that is loaded and initialized whenever the server starts and shuts down. Each of these classes must implement the `com.ibm.websphere.runtime.CustomService` interface. After you create a custom service, use the administrative console to configure that custom service for your application servers.

Configuring the JVM

As part of configuring an application server, you might define settings that enhance the way your operating system uses of the Java virtual machine (JVM).

Tuning application servers

The product contains interrelated components that must be harmoniously tuned to support the custom needs of your end-to-end e-business application.

Managing Object Request Brokers

An Object Request Broker (ORB) manages the interaction between clients and servers using the Internet InterORB Protocol (IIOP). There are several ways to manage an ORB. For example, you can use ORB custom property settings, or system property settings to configure an ORB, or you can provide objects during ORB initialization.

Introduction: Web servers

An application server works with a Web server to handle requests for dynamic content, such as servlets, from Web applications. A Web server uses a Web server plug-ins to establish and maintain persistent HTTP and HTTPS connections with an application server.

The Supported Hardware and Software Web page provides the most current information about supported Web servers.

Communicating with Web servers describes how to set up your Web server and Web server plug-in environment and how to create a Web server definition. The Web server definition associates a Web server with an application server. After you create a Web server definition, you can use the administrative console to perform the following functions for that Web server:

- Check the status of the Web server
- Generate a plug-in configuration file for that Web server.

You can not propagate a plug-in configuration file for a non-IBM HTTP Server. You must manually install an updated plug-in configuration file on that Web server.

After you set up your Web server and Web server plug-in, whenever you deploy a Web application, you must specify a Web server as the deployment target that serves as a router for requests to the Web application. The configuration settings in the plug-in configuration file (plugin-cfg.xml) for each Web server are based on the applications that are routed through that Web server. If the Web server plug-in configuration service is enabled, a Web server plug-in's configuration file is automatically regenerated whenever a new application is associated with that Web server.

Note: Before starting the Web server, make sure you are authorized to run any Application Response Measurement (ARM) agent associated with that Web server.

Refer to your Web server documentation for information on how to administer that Web server. For tips on tuning your Web server plug-in, see Web server plug-in tuning tips.

Introduction: Environment

Your WebSphere Application Server product environment includes Web server plug-ins, WebSphere Application Server variables, and other data objects. Configure values for settings in these categories using the Environment section of the administrative console.

Web servers

In the WebSphere Application Server product, an application server works with a Web server to handle requests for Web applications. The application Server and Web server communicate using a WebSphere HTTP plug-in for the Web server.

For more information, refer to “Introduction: Web servers” on page 24.

Variables

Variables come in many varieties. They are used to control settings and properties relating to the server environment. The three main types of variables that are important for you to understand are environment variables, WebSphere variables, and custom properties.

For more information, refer to “Introduction: Variables” on page 27.

Related concepts

“Introduction: Web servers” on page 24

An application server works with a Web server to handle requests for dynamic content, such as servlets, from Web applications. A Web server uses a Web server plug-ins to establish and maintain persistent HTTP and HTTPS connections with an application server.

“Introduction: Variables” on page 27

Variables come in many varieties. They are used to control settings and properties relating to the server environment. The three main types of variables that you should understand are environment variables, WebSphere variables, and custom properties.

What is new for scripted administration (wsadmin)

This topic highlights what is new or changed, for users who are going to customize, administer, monitor, and tune production server environments using the wsadmin tool.

Deprecated, stabilized, and removed features describes features that are being replaced or removed in this or future releases.

Improved administrative scripting features

Jython script library

The Jython script library provides a set of procedures to automate the most common application server administration functions. For example, you can use the script library to easily configure servers, applications, mail settings, resources, nodes, business-level applications, clusters, authorization groups, and more. You can run each script procedure individually, or combine several procedures to quickly develop new scripts.

For more information, see [Using the script library to automate the application serving environment](#).

Wildcard pattern usage for the AdminTask help and AdminConfig list commands

Use Java regular expression patterns and wildcard patterns with the `AdminTask.help('-commands')` command to query for command names and with `AdminConfig.list`, `AdminConfig.types` and `AdminConfig.listTemplates` functions to query for configuration object types.

For more information about using patterns with the `AdminTask` object, see [Obtaining online help using scripting](#).

Complex parameter type support

For more information about using patterns with the `AdminConfig` object, see [Commands for the AdminConfig object](#).

Use primitive and complex Java™ data types for the parameters of the `AdminTask` commands.

For more information, see [Data types for the AdminTask object](#).

AdminTask command support for configuring certificate authority (CA) clients

Use the `wsadmin` tool to manage certificate authority (CA) client configurations. Use the commands and parameters in the `CAClientCommands` group to create, modify, query, and remove connections to a third-party CA server.

For more information, see [CAClientCommands command group for the AdminTask object](#).

AdminTask command support for configuring security auditing

While security authentication and authorization ensures that users must have access to view protected resources, security auditing provides a mechanism to validate the integrity of a security computing environment. Security auditing collects and logs authentication, authorization, system management, security, and audit policy events in audit event records. You can analyze audit event records to determine possible security breaches, threats, attacks, and potential weaknesses in the security configuration of your environment. Enable security auditing in your environment.

For more information, see [Configuring security auditing using scripting](#).

AdminTask command support for configuring multiple security domains

Create multiple security configurations and assign them to different applications in WebSphere Application Server processes. By creating multiple security domains, you can configure different security attributes for both administrative and user applications within a cell environment. You can configure different applications to use different security configurations by assigning the servers or clusters or SIBuses that host these applications to the security domains. Only users assigned to the administrator role can configure multiple security domains.

For more information, see *Configuring multiple security domains using scripting*

AdminTask command support for configuring business-level applications

A business-level application is an administration model that provides the entire definition of an application as it makes sense to the business. A business-level application is a WebSphere® configuration artifact, similar to a server or cluster, that is stored in the product configuration repository.

For more information, see *Setting up business-level applications using scripting*

AdminTask command support for managing configurations using properties files

Manage your system configuration using properties files. Use the commands in the PropertiesBasedConfiguration group to copy configuration properties from one environment to another, troubleshoot configuration issues, and to apply one set of configuration properties across multiple profiles, nodes, cells, servers, or applications.

For more information, see *PropertiesBasedConfiguration command group for the AdminTask object*

AdminTask command support for configuring proxy servers

Use the commands and parameters in the ProxyManagement group to configure proxy servers for Web modules.

For more information, see *ProxyManagement command group for the AdminTask object*

Introduction: Variables

Variables come in many varieties. They are used to control settings and properties relating to the server environment. The three main types of variables that you should understand are environment variables, WebSphere variables, and custom properties.

Environment variables. Environment variables, also called *native environment variables*, are not specific to WebSphere Application Server and are defined by other elements, such as UNIX, Language Environment® (LE), or third-party vendors, among others. Some of the UNIX-specific native variables are LIBPATH and STEPLIB. These variables tend to be operating system-specific.

Environment variables can also be specified as an application server environment entry. To specify an environment variable as an environment entry, in the administrative console, click **Servers > Server Types > WebSphere application servers***server_name*. Then, under Server Infrastructure, click **Java process management > Process definition > Environment entries**.

WebSphere variables

WebSphere variables are name and value pairs that are used to provide settings for any of the string data type attributes contained in one of the XML formatted configuration files that reside in the product

repository. After a variable is defined, the value specified for the variable replaces the variable name whenever the variable name is encountered during configuration processing.

WebSphere variables can be used to configure:

- WebSphere Application Server path names, such as JAVA_HOME, and APP_INSTALL_ROOT
- Certain cell-wide customization values

To create or modify a WebSphere variable, in the administrative console click **Environment > WebSphere variables**.

A variable can apply to a node or a server.

How the variable is set determines its scope. If the variable is set:

- At the server level, it applies to the entire server.
- At the node level, it applies to all servers in the node, unless you set the same variable at the server level. In that case, for that server, the setting that is specified at the server level overrides the setting that is specified at the node level.

Custom properties

Custom properties are property settings meant for a specific functional component. Any configuration element can have a custom property. Common configuration elements are cell, node, server, Web container, and transaction service. A limited number of supported custom properties are available and these properties can be set in the administrative console using the custom properties link that is associated with the functional component.

For example, to set Web container custom properties, click **Servers > Server Types > WebSphere application servers > server_name**, and then, in the Container settings section, click **Web container > Custom properties**

Custom properties set from the Web container custom properties page apply to all transports that are associated with that Web container; custom properties set from one of the Web container transport chain or HTTP transport custom properties pages apply only to that specific HTTP transport chain or HTTP transport. If the same property is set on both the Web container page and either a transport chain or HTTP transport page, the settings on the transport chain or HTTP transport page override the settings that are defined for the Web container for that specific transport.

Note: You can only specify custom properties for an HTTP transport that is being used by an application server that is running on a Version 5.1.x node in a mixed cell environment.

Language versions offered by this product

This product is offered in several languages, as enabled by the operating platform on which the product is installed.

The following language versions are available:

- Brazilian Portuguese
- Chinese (Simplified)
- Chinese (Traditional)
- Czech
- English
- French
- German

- Hungarian
- Italian
- Japanese
- Korean
- Polish
- Russian
- Spanish

Chapter 2. Using the administrative clients

The product provides a variety of administrative clients for deploying and administering your applications and application serving environment, including configurations and logical administrative domains.

- “Using the administrative console”

The administrative console is a graphical, browser-based tool.

- Getting started with scripting

Scripting is a non-graphical alternative that you can use to configure and administer your applications and application serving environment. The WebSphere Application Server **wsadmin** tool provides the ability to run scripts. The wsadmin tool supports a full range of product administrative activities.

See also “Configuring Qshell to run WebSphere Application Server scripts” on page 148.

- Chapter 3, “Using Ant to automate tasks,” on page 213

To support using Apache Ant with Java Platform, Enterprise Edition (Java EE) applications running on IBM WebSphere Application Server, the product provides a copy of the Ant tool and a set of Ant tasks that extend the capabilities of Ant to include product-specific functions.

- “Using administrative programs (JMX)” on page 112

The product supports access to the administrative functions through a set of Java classes and methods, under the Java Management Extensions (JMX) specification. You can write a Java program that performs any of the administrative features of the other administrative clients. You also can extend the basic product administrative system to include your own managed resources.

- “Using command line tools” on page 147

Several command-line tools are available that you can use to start, stop, and monitor WebSphere server processes and nodes. These tools work on local servers and nodes only. They cannot operate on a remote server or node.

See also “Configuring Qshell to run WebSphere Application Server scripts” on page 148.

Using the administrative console

You can install, start, and access the administrative console. You can also specify console preferences and access help.

About this task

The administrative console is a Web-based tool that you use to manage the product. The administrative console supports a full range of product administrative activities.

Application servers and administrative agents can have their own administrative consoles. The steps in this task apply to these consoles.

1. Optionally install the administrative console through the wsadmin command.
2. Start the server for the appropriate administrative console.

Issue the startServer command for an application server or an administrative agent server. The administrative console application starts automatically when you start the server to which the administrative console belongs.

3. Access the administrative console.
4. Specify console preferences.
5. Access help.

Installing and uninstalling the administrative console

You can install the administrative console during profile creation or after you create a profile. You can uninstall any administrative console that you install. To install an administrative console after profile

creation, or to uninstall the administrative console, use the **wsadmin** command. This topic discusses how to use the **wsadmin** command to install and uninstall the administrative console.

Before you begin

If you install the administrative console through the **wsadmin** command, a profile that does not have an administrative console installed must exist.

About this task

Run the `deployConsole` script on the **wsadmin** command whenever you want to uninstall the administrative console, or whenever you want to install the administrative console to a profile that does not have an administrative console installed.

You can run the script in either connected or disconnected mode.

Application servers and administrative agents can have their own administrative consoles. The steps in this task apply to these consoles.

The usual security restrictions for the **wsadmin** command apply to this script. In connected mode, the user must authenticate if security is enabled.

The `deployConsole.py` script is located in the `profile_root/bin` directory.

- To install the administrative console, issue the following command:

```
wsadmin -f deployConsole.py install
```
- To uninstall the administrative console, issue the following command:

```
wsadmin -f deployConsole.py remove
```

Results

The administrative console is installed or uninstalled, depending on whether you specified the `install` or `remove` option.

Starting and logging off the administrative console

This topic describes how to set up the administrative console environment, to access the administrative console, and to log out of the administrative console.

Before you begin

To access the administrative console, you must first install WebSphere Application Server and the administrative console.

About this task

You start the administrative console, access the console through a Web browser, and then log into the administrative console. After you finish working in the administrative console, save your work and log out.

1. Start the desired administrative console by starting the server process that runs the console application.

You can start an application server or an administrative agent server.

Check the `SystemOut.log` file of the server that runs the console application to verify that the console application starts successfully. If the console application starts successfully, you see the `WSVR0221I: Application started: isclite` message.

If you cannot start the administrative console because the console port conflicts with an application that is already running on the machine, use the `chgwassvr` script command to change the port number.

Read about changing the ports associated with an application server for more information. Alternatively, shut down the other application that uses the conflicting port before starting the WebSphere Application Server product.

2. Access the administrative console.

- a. Enable cookies in the Web browser that you use to access the administrative console.
- b. Enable JavaScript™.

Enablement of JavaScript is required. You must enable JavaScript so that all the features of the administrative console are available.

- c. In the same Web browser, type `http://your_fully_qualified_server_name:port_number/ibm/console`, where *your_fully_qualified_server_name* is the fully qualified host name for the machine that contains the administrative server, and *port_number* is the administrative console port number. When the administrative console is on the local machine, *your_fully_qualified_server_name* can be `localhost` unless security is enabled. On Windows platforms, use the actual host name if `localhost` is not recognized. If security is enabled, your request is redirected to `https://your_fully_qualified_server_name:port_number/ibm/console`, where *your_fully_qualified_server_name* is the fully qualified host name for the machine that contains the administrative server, and *port_number* is the administrative console secure port number.

The administrative consoles for the application server and the administrative agent use 9060 as the default port number for an unsecure administrative console and 9043 as the default port number for a secure administrative console. Each new administrative console that you deploy during profile creation is assigned a new unsecure port number, and, if you enable security during profile creation, a new secure port number. Whichever unsecure administrative console you create first is assigned port 9060, as long as the port is available. Likewise, whichever secure administrative console you create first is assigned port 9043, as long as the port is available.

For a listing of supported Web browsers, see WebSphere Application Server system requirements.

- d. Wait for the administrative console to load into the browser.

A login page displays after the administrative console starts.

3. Log into the administrative console.

The administrative console can be for an application server or an administrative agent.

- a. If you are logging into the administrative console for the administrative agent, and you have registered at least one node with the administrative agent, select the node to administer, and click **Continue**.

The node can be the administrative agent node or a node for one of the application servers registered to the administrative agent. After you select a node, the login procedure is the same as that for the other server types, and for administrative agents with no nodes registered.

- b. Enter your user name or user ID.

The user ID lasts only for the duration of the session for which it is used to log in.

Changes made to server configurations are saved to the user ID. Server configurations also are saved to the user ID if a session timeout occurs.

See the *Administering applications and their environment* PDF for more information on server configurations.

If you enter an ID that is already in use and in session, you are prompted to do one of the following actions:

- Log out the other user with the same user ID. You can recover changes made during the other user's session.
 - Return to the login page and enter a different user ID.
- c. If the console is secure, you must also enter a password for the user name. The console is secure if someone has taken the following actions for the console:
 - Specified security user IDs and passwords
 - Enabled global security

See the *Securing applications and their environment* PDF for more information.

d. Click **OK**.

4. Log off the administrative console. Click **System administration > Save changes to master repository > Save** to save work. Then click **Logout** to exit the console.

If you close the browser before saving your work, you can recover any unsaved changes the next time that you log in under the same user ID.

Results

You have set up the administrative console environment, accessed the administrative console, and logged out of the administrative console.

What to do next

Use the administrative console to manage the product.

Logging in

Enter your user ID and password to access the console.

To access the console, enter your **User ID** and **Password** and then click **Log in**. The password is required only if security is enabled. In environments that use the administrative agent to administer multiple application server nodes, select whether to log in to the administrative agent or one of its registered profiles.

After you are logged in, be sure to use the **Logout** link in the console toolbar when you are finished using the console and to prevent unauthorized access. If there is no activity during this login session for an extended period of time, the session expires and you must login again to access the console. The administrator can change the session timeout. The default is set to 30 minutes.

If the user ID that you provide is already logged in at a different location, you are prompted to choose between logging out from the other location or returning to the login page. If you log out the user from the other location, you might be prompted to recover unsaved changes made by that user.

If you have one or more different standalone servers running on the same machine and wish to administer them concurrently from the same or a different machine then you must:

1. Ensure that each server uses a unique value for its admin console port.
2. Run a separate web browser process for each admin console that you wish to access concurrently.

Related tasks

“Starting and logging off the administrative console” on page 32

This topic describes how to set up the administrative console environment, to access the administrative console, and to log out of the administrative console.

Save changes to the master configuration

Use this topic to update the master repository with your administrative console changes, to discard your administrative console changes and continue working with the master repository, or to continue working with your administrative console changes that are not saved to the master repository.

Until you save changes to the master repository, the administrative console uses a local workspace to track your changes.

Total changed documents: Specifies the total number of documents that you changed for your session, but that are not saved to the master repository. By clicking the +/- toggle key, you can see additional information about the changed documents:

- **Changed items**

When you change your local configuration, each path and configuration file that you can apply the update to in the master repository is displayed in the list.

- **Status**

The status can contain the following options:

- **Added:** If you save your changes to the master repository, a new configuration file is created on the indicated path.
- **Updated:** If you save your changes to the master repository, an existing configuration file is updated on the indicated path.
- **Deleted:** If you save your changes to the master repository, an existing configuration file is deleted on the indicated path.

Synchronize changes with nodes: Specifies whether you want to force node synchronization at the time that you save your changes to the master repository, rather than when node synchronization normally occurs.

Save conflict: Specifies that another user changed some configuration information since you began making changes. You can either click **Save** to overwrite the other user information, or **Discard** to discard your changes and keep the changes that the other user made.

Administrative console buttons

This page describes the button choices that are available on various pages of the administrative console, depending on which product features you enable.



- **Select all items.** Selects each resource that is listed on the administrative console panel, in preparation for performing an action against the selected resources.
- **Deselect all items.** Removes all the listed resources from each selection so that no action is performed against any of the resources.
- **Show filter function.** Produces a dialog box for specifying the resources to view in the table on this administrative console page.
- **Hide filter function.** Hides the dialog box for specifying the resources to view in the table on this administrative console page.

When you produce the dialog box, select the column to filter and enter the filter criteria.

Column to filter

Select the column to filter from the drop-down list. When you apply the filter, only those items in the selected column that meet the filter criteria are displayed.

For example, select **Names** to enter criteria by which to filter application server names.

Filter criteria

Enter a string that must be found in the name of a collection entry to qualify the entry to display in the collection table. The string can contain percent sign (%), asterisk (*), or question mark (?) symbols as wildcard characters. For example, enter *App* to find any application server whose name contains the string App.

Prefix each of the following characters () ^ * % { } \ + \$ with a backslash (\) so that the regular expression engine performing the search correctly matches the search criteria. For example, to search for all Java DataBase Connectivity (JDBC) providers containing (XA) in the provider name, specify the following string:

```
*\ (XA\)
```

- **Clear filter value.** Clears your filter changes and restores the most recently saved values.
- **Abort.** Stops a transaction that is not yet in the prepared state. All operations that the transaction completed are undone.
- **Activate.** Activates a group member.
- **Add.** Adds the selected or typed item to a list, or produces a dialog for adding an item to a list.
- **Apply.** Saves your changes to a page without exiting the page.

- **Back.** Displays the previous page or item in a sequence. The administrative console does not support using the Back and Forward options of a browser, which can cause intermittent problems. Use Back or Cancel on the administrative console panels instead.
- **Browse.** Opens a dialog that enables you to look for a file on your system.
- **Cancel.** Exits the current page or dialog, discarding unsaved changes. The administrative console does not support using the Back and Forward options of a browser, which can cause intermittent problems. Use Cancel on the administrative console panels instead.
- **Change.** In the context of security, you can search the user registry for a user ID for an application to run under. In the context of container properties, you can change the data source that the container is using.
- **Clear.** Clears your changes and restores the most recently saved values.
- **Clear selections.** Clears any selected cells in the tables on this tabbed page.
- **Close.** Exits the dialog.
- **Collapse all.** Collapses all the expanded items.
- **Commit.** Releases all locks that are held by a prepared transaction and forces the transaction to commit.
- **Copy.** Creates copies of the selected application servers.
- **Create.** Saves your changes to all the tabbed pages in a dialog and exits the dialog.
- **Create tables.** Develops scheduler database tables.
- **Deactivate.** Deactivates a group member. The group member must be in the active state to be deactivated. The deactivate option causes the group member to move to the idle state. The group policy overrides which members are activated and deactivated for a group. The policy is enforced for every member state change. If the deactivate option conflicts with the group policy, the policy resets who is the active member of the group.
- **Delete.** Removes the selected instance.
- **Details.** Shows the details about a transaction.
- **Disable.**

Group or group members:

Disables a group or group member. When you disable a group or group member, the active group or group member is first deactivated. If the deactivate option is successful, the group or group member moves to the disable state. A disabled group or group member cannot be activated.

Java Management Extensions (JMX) connectors:

Disables a connector.

- **Disable Auto Start.** Requires you to start the application manually.
- **Discard.** Discards your local changes instead of saving them to the master configuration.
- **Done.** Saves your changes to all the tabbed pages in a dialog and exits the dialog.
- **Down.** Moves through a list.
- **Drop tables.** Removes scheduler database tables.
- **Dump.** Activates a dump of a traced application server.
- **Edit.** Lets you edit the selected item in a list, or produces a dialog box for editing the item.
- **Enable.**

Group or group members:

Enables a group or a group member.

Java Management Extensions (JMX) connectors:

Enables a connector.

- **Enable Auto Start.** Starts an application automatically when the server on which the application resides starts.
- **Expand all.** Expands all the collapsed items.
- **Export.** Accesses a page for exporting enterprise archive (EAR) files for an enterprise application.
- **Export DDL.** Accesses a page for exporting data definition language (DDL) files for an enterprise application.
- **Export Keys.** Exports Lightweight Third-Party Authentication (LTPA) keys to other domains.
- **Export route table.** Exports the route table information for a selected cluster to a binary file in the configuration.
- **Filter.** Produces a dialog box for specifying the resources to view in the tables on this tabbed page.

- **Finish.** Forces a transaction to finish, regardless of whether its outcome has been reported to all participating applications.
- **First.** Displays the first record in a series of records.
- **Generate keys.** Generates new LTPA keys. When security is turned on for the first time with LTPA as the authentication mechanism, LTPA keys are automatically generated with the password entered in the panel. To generate new keys, use this option after the server is up with security turned on. Clicking this option generates the keys and propagates them to all active servers (cell, node, and application servers). The new keys can be used to encrypt and decrypt the LTPA tokens. Click **Save** on the console taskbar to save the new keys and the password in the repository.
- **Immediate stop.** Stops the server, but bypasses the normal server quiesce process that supports in-flight requests to complete before shutting down the entire server process. This shutdown mode is faster than the normal server stop processing, but some application clients can receive exceptions.
- **Import keys.** Imports new LTPA keys from other domains. To support single sign-on (SSO) in WebSphere Application Server across multiple WebSphere domains (cells), share LTPA keys and a password among the domains. After exporting the keys from one of the cells into a file, click this option to import the keys into all the active servers (cell, node, and application servers). The new keys can be used to encrypt and decrypt the LTPA token. Click **Save** on the console taskbar to save the new keys and the password in the repository.
- **Install.** Displays the Preparing for application installation page, which you use to deploy an application, an enterprise bean, or a Web component onto an application server.
- **Install RAR.** Opens a dialog that is used to install a Java 2 Platform, Enterprise Edition Connector Architecture (JCA) connector and to create a resource adapter.
- **Manage state.** Displays a list of MBeans that corresponds to your previous selection of data source or connection factory configurations. You can apply JCA lifecycle management operations to these MBeans to control the runtime status of the corresponding resources.
- **Manage transactions.** Displays a list of active transactions running on a server. You can forcibly finish any transaction that has stopped processing because a transactional resource is not available.
- **Modify.** Opens a dialog that is used to change a specification.
- **Move.** Moves the selected application servers to a different location in the administrative cell. When prompted, specify the target location.
- **Move down.** Moves downward through a list.
- **Move up.** Moves upward through a list.
- **New.** Displays a page that you use to define a new instance. For example, clicking **New** on the Application Servers page displays a page on which you can configure a new application server.
- **Next.** Displays the next page, frame, or item in a sequence.
- **OK.** Saves your changes to the local configuration and exits the page.
- **Pause.** In the context of JCA lifecycle management, stops all outbound communication that is conducted through a resource on a specified server to a backend.
- **Ping.** Attempts to contact selected application servers.
- **Previous.** Displays the previous page, frame, or item in a sequence.
- **Quit.** Exits a dialog box and discards any unsaved changes.
- **Reference shared libraries.** Opens the collection of shared library references available for use by your application or module. If no references are available, a message is displayed stating that there are no references.
- **Refresh.** Refreshes the view of data for instances that are currently listed on this tabbed page.
- **Remove.** Deletes the selected item.
- **Remove file.** Removes the specified file from the selected application or module.
- **Reset.** Clears your changes on the tab or page and restores the most recently saved values.
- **Resume.** In the context of JCA lifecycle management, restarts the activity of a data source or a connection factory that was paused by a previous JCA lifecycle management operation.
- **Retrieve new.** Retrieves a new record.
- **Save.** Saves the changes in your local configuration to the master configuration.
- **Select.** For resource analysis, lets you select a scope in which to monitor resources.
- **Set.** Saves your changes to settings in a dialog.
- **Settings.** Displays a dialog for editing servlet-related resource settings.
- **Settings in use.** Displays a dialog showing the settings in use.

- **Start.** In the context of application servers, starts selected application servers. In the context of data collection, starts collecting data for the tables on this tabbed page.
- **Stop.** In the context of server components such as application servers, stops the selected server components. In the context of a data collection, stops collecting data for the tables on a tabbed page.
- **Terminate.** Deletes the Application Server process or another process that cannot be stopped by the **Stop** or **Immediate Stop** commands. Some application clients can receive exceptions. Always attempt an immediate stop before using this option.
- **Test connection.** After you define and save a data source, you can select this option to ensure that the parameters in the data source definition are correct. On the Collection panel, you can select multiple data sources and test them simultaneously.
- **Uninstall.** Deletes a deployed application from the WebSphere Application Server configuration repository. Also deletes application binary files from the file system.
- **Update.** For applications, replaces an application that is deployed on a server with an updated application. As part of the updating, you might need to complete steps on the Preparing for application installation and Update application pages.
- **Update resource list.** Updates the data on a table. Discovers and adds new instances to the table.
- **Verify tables.** Validates the mapping between the table names, scheduler resource, and data sources.
- **View.** Opens a dialog on a file.

Related tasks

“Using the administrative console” on page 31

You can install, start, and access the administrative console. You can also specify console preferences and access help.

Administrative console page features

This topic provides information about the basic elements of an administrative console page, such as the various tabs.

Administrative console pages are arranged in a few basic patterns. Understanding their layout and behavior can help you use them more easily.

Collection pages

Use collection pages to manage a collection of existing administrative objects. A collection page typically contains one or more of the following elements:

Scope Scope is described in Administrative console scope settings.

Preferences

Preferences are described in Administrative console preference settings.

Table of existing objects

The table displays existing administrative objects of the type specified by the collection page. The table columns summarize the values of the key settings for these objects. If no objects exist yet, an empty table is displayed. Use the available options to create a new object.

Buttons for performing actions

The available actions are described on the Administrative console buttons help panel. In most cases, you need to select one or more of the objects in the table, then click an action. The action is applied to the selected objects.

Sort toggle buttons

The column headings in the table are followed by icons for sort ascending (^) and sort descending (v). By default, items such as names are sorted in descending order (alphabetically). To enable another sorting order, click the icons for the column that you want to sort.

Detail pages

Use detail pages to configure specific administrative objects, such as an application server. A detail page typically contains one or more of the following elements:

Configuration tabbed page

This tabbed page is for modifying the configuration of an administrative object. Each configuration page has a set of general properties that is specific to the administrative object. Other sets of properties display on the page, but vary depending on the administrative object.

Runtime tabbed page

This tabbed page displays the configuration that is currently in use for the administrative object. The object is read-only in most cases. Some detail pages do not have runtime tabs.

If you can edit runtime properties, these properties directly affect the current runtime environment, but are not preserved when that environment is stopped.

Local Topology tabbed page

This tabbed page displays the topology that is currently in use for the administrative object. View the topology by expanding and collapsing the different levels of the topology. Some detail pages do not have local topology tabs.

Buttons for performing actions

Buttons to perform specific actions display on the configuration tabbed page and the Runtime tabbed page. The displayed buttons vary based on the administrative object. The available buttons are described on the Administrative console buttons help panel.

Wizard pages

Use wizard pages to complete a configuration process comprised of several steps. Be aware that wizards show or hide certain steps depending on the characteristics of the specific object that you are configuring.

Related tasks

“Using the administrative console” on page 31

You can install, start, and access the administrative console. You can also specify console preferences and access help.

Related reference

“Administrative console scope settings” on page 47

Use this page to specify the level at which a resource is visible on the administrative console panel. By changing the value for Scope, you see only the resources that are defined at that scope. The contents of the collection table might change. For WebSphere Application Server and Express, a resource can be visible in the administrative console collection table at the cell, node, or server scope.

“Administrative console preference settings” on page 46

Use the preference settings to specify how you want information to display on an administrative console panel. The preference settings vary from one administrative console panel to another.

Console layout

This topic describes the layout of the user interface for Integrated Solutions Console.

See Navigating the console for instructions on how to use the console controls.

Banner

Displays a common image across all Integrated Solutions Console installations. The banner includes a greeting to the user who is logged in and links to log out of the console and to open console help.

Navigation tree

Lists the tasks available in the console. Tasks are grouped into organizational nodes that represent categories of tasks, for example, Servers, or Applications. The organizational nodes can be nested in multiple levels.

The tasks shown are only those for which the user has access. When you click a task in the navigation, a page is displayed in the work area containing one or more modules for completing the task. Use the View selection list at the top of the navigation area to modify the list of tasks according to your preferences. You can organize the tasks as follows:

All tasks

This shows all tasks in the console. Tasks are grouped into organizational nodes, for example, Guided activities, Servers, or Applications.

My tasks

This shows only the tasks that you have added to the view. This list is initially empty, but provides a link to the **My Tasks** module. Use **My Tasks** to add and remove from the My Tasks list in the navigation.

Product selection

Selecting a product name shows only the tasks for that particular product, for example, WebSphere Application Server.

Work area

When you launch a page, the content of the page is displayed in the work area. If you have not launched any pages, the Welcome page is displayed in the work area. A page contains one or more console modules that are used to perform operations. Each console module has its own navigation controls. Some pages include a control to close the page and return to the Welcome page.

Console navigation

This topic describes how to navigate pages and tasks in the Integrated Solutions Console.



- Launching pages from the navigation tree
- Filtering tasks in the navigation
- Using the title bar controls
- Accessing help
- Using the console help controls

Be sure you understand the terms and information in the “Console layout” on page 39 before reading this section.

Launching pages from the navigation tree

The console navigation provides a hierarchical view of all of the *tasks* available in the console. A task is a page in the work area consisting of one or more console modules. All of the modules on the page are provided to start and complete the task. To open a task, simply click the task name in the navigation. The task is opened in a new page in the work area.

The following table describes the controls for the console navigation tree and entries in the tree.









Icon	Function
	Represents an organizational node in the navigation tree that contains pages or other navigation nodes. Click the icon to expand the node.
	Closes an organizational node.

Filtering tasks in the navigation

When you first access the console, all tasks to which you have access are displayed in the navigation. Use the view menu at the top of the navigation to filter the list of tasks by product. Or, you can create a customized list by selecting My tasks from the **View** menu. For instructions on creating and managing your custom list of tasks, see My tasks.

Using the title bar controls

Each page contains one or more Web applications or *console modules*. A console module enables you to perform an operation, such as displaying a list or stopping a managed system. The title and the controls for the module are displayed on the title bar. Depending on the functions supported by the module, the following icons might be displayed on the title bar:

- The  icon is displayed if the module allows you to edit settings for the portlet. For example, a module that retrieves performance data could permit you to specify the server to be analyzed. When you click the icon, an edit screen is displayed. Click the  icon to return to the previous screen.
- The  icon allows you to return to the previous screen.
- The  icon is displayed if help is available for the module. When you click the icon, the help is displayed in a separate browser window.
- The  icon allows you to minimize the module view. When you click the icon, only the title bar is visible on the page. Click the  icon to return to the maximize state.
- The  icon allows you to maximize a module view. When you click the icon, the full portlet view is visible on the page. Click the  icon to return to the minimize state.

In addition to the controls on the title bar, a module can include controls for other actions, such as a button to submit input. Some modules have controls that launch other modules. If a module launches another module, the newly launched module is displayed on a new page.


Accessing help

Help is available for the entire console or for a specific module in the console.




To access console help, perform the following steps:








1. Click **Help** on the console toolbar. The Help is displayed in a separate browser window.
2. In the help navigation tree, click the help set you want to view. For example, click **Console help** to view topics that provide helpful information for new console users. Use the console help controls as needed.

To access help for a module on a page, perform the following steps:

1. On the title bar for the module, click the  icon. That icon is displayed only if help is available for the module. The help is displayed in a separate browser window.
2. Close the help window when you are finished viewing it.

Using the console help controls

Icon	Function
	Use these controls to navigate the list of pages you have viewed. Click  to return to the previous help topic that was displayed. Click  to move forward in the history list.

Icon	Function
	Click either of these icons to synchronize the navigation tree with the current topic. The current topic will be highlighted in the navigation tree. This function is useful if you followed links from one help topic to other topics and you want to determine where the current topic is listed in the help navigation tree.
	Permits you to add the current page to your browser favorites list or bookmarks.
	Displays a window for printing the help topic that is displayed.
	Maximizes the target view. This control is available for the Table of Contents view, the Search Results view, and the topic display area.
	Restores a maximized view to its normal size.
	Changes the view to the Search Results view. To search all of the help topics, type a word or words in the Search field. Enclose a phrase within double quotes. You can use Boolean operators (such as OR) in the search string. To limit the scope of the search, click Search scope . Click GO to start the search. A list of topics that contain the target strings are displayed in the results frame.
	Changes from the Search Results view to the Table of Contents view.

Administrative console browser support

Several Web browsers are supported for use with Integrated Solutions Console.

The following Web browsers are supported for use with Integrated Solutions Console:

- Firefox V2.0.3.0
- Firefox V1.5, for AIX V6.1 only
- Microsoft® Internet Explorer V7.0
- Microsoft Internet Explorer V6.0 SP2

Note: Using the browser's back button with the console can produce unexpected results and is not supported. Use the controls and links provided in the console to navigate between pages and applications.

Related tasks

“Using the administrative console” on page 31

You can install, start, and access the administrative console. You can also specify console preferences and access help.

Console accessibility

There are many accessibility features built into Integrated Solutions Console.

- Accessibility features
- Navigating the console by using the keyboard
- Navigating help by using the keyboard

Accessibility features

The Integrated Solutions Console has the following accessibility features:

- The following features are for vision-impaired users:
 - Can be operated by using only the keyboard
 - Communicates all information independent of color
 - Supports the attachment of alternate output devices
 - Provides help information in an accessible format
- The following features are for users who have mobility impairments or limited use of their hands:
 - Allows the user to request more time to complete timed responses
 - Can be operated by using only the keyboard
 - Supports the attachment of alternative input and output devices
- The following features are for the deaf and hard of hearing users:
 - Supports alternatives to audio information
 - Supports adjustable volume control
- The console does not flash the screen at rates that could induce epileptic seizures.

The help system for Integrated Solutions Console has the following accessibility features:

- Uses the accessibility support enabled by the browser that is used to display the help
- Enables navigation by using the keyboard

Navigating the console by using the keyboard

To move through the controls on a particular page, use the Tab key.

To click a link or control on a page using the keyboard, navigate to the link or control and press Enter.

To change the navigation view using the keyboard, follow these steps.

1. Navigate to the **View** selection list using the Tab key.
2. Use the up and down arrows to change the value of the selection list.
3. Press Enter. The tasks displayed in the navigation are changed according to your selection.

Navigating help by using the keyboard

Use the following key combinations to navigate the help system by keyboard:

- To bring the Topic pane (the right hand side) into focus, press Alt+K, and then press Tab.
- In the Topic pane, to go to the next link, press Tab. To go to the previous link, press Shift+Tab.
- To go directly to the Search Results view in the left hand side, press Alt+R, and then press Enter or Up arrow to enter the view.
- To go directly to the Navigation (Table of Contents) view in the left hand side, press Alt+C, and then press Enter or Up arrow to enter the view.
- To navigate your browser history, press Alt+Left arrow to go back. If you have navigated back to a previously view page, you can use Alt+Right arrow to navigate forward again.
- To expand and collapse a node in the navigation tree, tab to the + or - image next to it to bring the image into focus, and then press the Right or Left arrows.
- To go to the next frame in the help system, press F6. To go to the previous frame in the help system, press Shift+F6.
- In the navigation, to move to the next topic node, press the Down arrow or Tab. To move to the previous topic node, press the Up arrow or Shift+Tab.

- To go to the next link, button, or topic node from inside a view, press Tab.
- To scroll all the way up or down in a frame, press Home or End, respectively.
- To print the active pane, press Ctrl+P.
- To move to the search entry field, press Alt+S.

Welcome

Display products that are installed that use the Integrated Solutions Console for administrative tasks.

The Welcome page displays the products that are installed that use the Integrated Solutions Console for administrative tasks. The page lists the product name and version number. If provided by the product, you can click the product name to display a page that provides more information about the product.

My tasks

Create and edit a task view in the console navigation.

Use My tasks to create and edit a list of tasks to view in the console navigation. A task includes a page that contains one or more Web applications, or *console modules*, that are used to complete that task. When you first access the console, all tasks to which you have access are displayed in the navigation. My tasks is especially useful to customize the navigation to show only the tasks you use most often. After you customize your tasks, My Tasks is initially displayed each time you log in to the console.

Follow these general steps to customize your task list in the navigation.

1. Select My tasks from the **View** selection list in the navigation. If you have never used My tasks before, you must click **Add tasks** to open it.
2. Use the checkboxes to select and deselect tasks from the My tasks navigation.
3. To save your changes, click **Apply**.
4. To cancel your changes, click **Reset**.

After applying your selections, your customized task list is displayed in the navigation.

Use the following buttons to customize your task selections.

Apply Saves the current selections.

Reset Backtracks all changes to the selections that were set since the last time My tasks was applied. This is useful if you need to cancel your changes.

Select All

Checks every task.

Deselect All

Unchecks every task.

Expand All

Expands each node in the display and reveals all subtasks in the navigation.

Collapse All

Collapses each node in the display so that only the top level nodes are displayed.

Specifying console preferences

Use this topic to customize how much data displays on an administrative console panel.

About this task

Throughout the administrative console are pages that have Preferences fields, Scope fields, and Filter radio buttons. By selecting these fields and radio buttons you can customize how much data is shown.

For example, examine the Preferences field for the Administrative authorization groups page:

1. Go to the navigation tree of the administrative console and click **Security > Administrative authorization groups**.
2. Expand **Preferences**.
3. For the **Maximum rows** field, specify the maximum number of rows to display when the collection is large. The default is 20. Rows that exceed the maximum number display on subsequent pages.
4. Select **Retain filter criteria** if you want to retain the last filter criteria that is entered in the filter function. When you return to the Applications page, the page initially uses the retained filter criteria to display the collection of applications in the table following the preferences. Otherwise, clear **Retain filter criteria** and the last filter criteria is not retained.
5. Click **Apply** to apply your selections or click **Reset** to return to the default values. The default is not to enable (not have a check mark beside) **Retain filter criteria**.

Results

Other pages have similar fields and radio buttons that you can use to specify console preferences. While Preferences fields, Scope fields, and Filter buttons control how much data is shown in the console, the **Preferences** option controls general behavior of the console. Click **System Administration > Console preferences** to view the Preferences page.

Console preferences settings

Use the Console Preferences page to specify how you want features of the administrative console workspace to behave.

To view this administrative console page, click **System administration > Console preferences**.

Turn on workspace automatic refresh:

Specifies whether you want the administrative console workspace to refresh automatically after the administrative configuration changes.

The default is for the workspace to refresh automatically. If you delete a WebSphere variable, for example, the WebSphere variables page refreshes automatically and shows the updated list of WebSphere variables in the WebSphere variables collection.

Specifying that the workspace not refresh automatically means that you must access a page again by clicking the console navigation tree or links on collection pages to see the changes that are made to the administrative configuration.

Default true (selected)

No confirmation on workspace discard:

Specifies whether the confirmation dialog is displayed after a request is received to discard the workspace. The default is to display confirmation dialogs.

Default false (cleared)

Use default scope:

Specifies whether the default scope is the administrative console node.

All scopes is the default unless you enable the Use default scope setting to make the administrative console node the default. Whatever the default is the first time that you view a console panel that has

scope settings, that is the default for the panel on subsequent visits that you make to the panel. The default for the panel does not change even if you modify the Use default scope setting.

Default false (cleared)

Show the help portlet: Specifies whether the help portlet on the right of the console displays.

Default true (selected)

Enable command assistance notifications: Specifies whether to send Java Management Extensions (JMX) notifications that contain command assistance data from the administrative console. Enablement of the notifications allows integration with product tools such as the Toolkit Jython editor for WebSphere Application Server. Enablement of this option is recommended for non-production environments only.

Default false (cleared)

Log command assistance commands: Specifies whether to log all the command assistance wsadmin data to a file. This file is saved to `${LOG_ROOT}/server/commandAssistanceJythonCommands_ user name.log`:

- *server* is the server process where the console runs, such as server1 or adminagent.
- *user name* is the administrative console user name.
- When you manage a profile using an administrative agent, the command assistance log is put in the location of the profile that the administrative agent is managing. The `${LOG_ROOT}` variable defines the profile location.

Occasionally clean out the file to manage its growth.

Default false (cleared)

Administrative console preference settings

Use the preference settings to specify how you want information to display on an administrative console panel. The preference settings vary from one administrative console panel to another.

Related tasks

“Specifying console preferences” on page 44

Use this topic to customize how much data displays on an administrative console panel.

Maximum rows: Indicates the maximum number of rows to display per page when the collection is large.

Filter history: Indicates whether to use the same filter criteria to display this page the next time that you visit.

Select the **Retain filter criteria** check box to retain the last filter criteria entered. When you return to the page, retained filter criteria control the application collection that is displayed in the table.

Show resources in the scope hierarchy: Select the check box if you want to display the resources in the hierarchy for a particular scope.

The hierarchy is:

- cell > node > server
- cell > cluster

For example, if you select a node scope, all node scope resources and all cell scope resources display for the node.

This preference is available for resource factory panels only.

Show built-in resources: Select the check box if you want to display resources that are pre-defined to support certain internal components of the product. For example, the product includes built-in configurations of a Cloudscape® JDBC provider and a data source to support the Universal Description, Discovery and Integration Protocol (UDDI) registry for Web services.

Show confirmation for stop command: Select the check box if you want a confirmation that the **stop** command is successful.

Show confirmation for immediate stop command: Select the check box if you want a confirmation that the **immediate stop** command is successful.

Show confirmation for terminate command: Select the check box if you want a confirmation that the **terminate** command is successful.

Show resources at one authorizing group level only: Specifies the authorization group level used to filter the resources in the table. Only those roles that apply to your ID can display in the table. Valid values are All Roles, Administrator, Deployer (for application collection panels only), Operator, Configurator, and Monitor. If All Roles is selected, then all the resources that you are authorized to view are displayed in the table grouped by role. Otherwise, the resources for the role selected display in the table.

Hide system queues: Select the box to hide messaging engine system queue points.

Include cluster members in the collection: Select the check box if you want the collection to include application servers that belong to a server cluster.

Show confirmation for update runtime command: Select the check box to enable the confirmation panel for the update runtime command button.

Show items at the following authorization group level: Select from the list the authorization group level that will be used to filter the items in the table.

Show all data source properties: By default, this panel does not list the custom properties that are configurable by different administrative console pages. Select the check box to show all data source custom properties, including properties required by the data source and configuration properties. Selecting this option does not affect what is displayed on other panels.

Administrative console scope settings

Use this page to specify the level at which a resource is visible on the administrative console panel. By changing the value for Scope, you see only the resources that are defined at that scope. The contents of the collection table might change. For WebSphere Application Server and Express, a resource can be visible in the administrative console collection table at the cell, node, or server scope.

For WebSphere Application Server and Express, the console displays a drop-down list of all the scopes available, which is three. To change the scope, select any item from the drop-down list.

All scopes is the default unless you enable the Use default scope setting on the Console preferences panel to make the administrative console node the default. Whatever the default is the first time that you view a console panel that has scope settings, that is the default for the panel on subsequent visits that you make to the panel. The default for the panel does not change even if you modify the Use default scope setting.

You cannot select All scopes to create a new resource. You must select one of the available scopes from the drop down list to create a new resource.

You always create resources at the current scope that is selected in the administrative console panel, even though the resources might be visible at more than one scope.

Resources such as Java Database Connectivity (JDBC) providers, namespace bindings, or shared libraries can be defined at multiple scopes. Resources that are defined at more specific scopes override duplicate resources that are defined at more general scopes:

- The application scope has precedence over all the scopes.
- For WebSphere Application Server and Express, the server scope has precedence over the node and cell scopes.
- The node scope has precedence over the cell scope.

Despite the scope of a defined resource, the resource properties apply at an individual server level only. For example, if you define the scope of a data source at the cell level, all the users in that cell can look up and use that data source, which is unique within that cell. However, resource property settings are local to each server in the cell. For example, if you define the maximum connections as 10, then each server in that cell can have 10 connections.

The cell scope is the most general scope and does not override any other scope. The recommendation is that you generally specify a more specific scope than the cell scope. When you define a resource at a more specific scope, you provide greater isolation for the resource. When you define a resource at a more general scope, you provide less isolation. Greater exposure to cross-application conflicts occur for a resource that you define at a more general scope.

Cell Limits the visibility to all servers on the named cell. The resource factories within the cell scope are:

- Defined for all servers within this cell
- For WebSphere Application Server and Express, overridden by any resource factories that are defined within application, server, and node scopes that are in this cell and have the same Java Naming and Directory Interface (JNDI) name

The resource providers that are required by the resource factories must be installed on every node within the cell before applications can bind or use them.

Node Limits the visibility to all the servers on the named node. The node scope is the default scope for most resource types. The resource factories that are defined within the node scope:

- Are available for servers on this node to use
- Override any resource factories that have the same JNDI name defined within the cell scope

The resource factories that are defined within the cell scope are available for servers on this node to use, in addition to the resource factories that are defined within this node scope.

Server

Limits the visibility to the named server. The server scope is the most specific scope for defining resources. The resource factories that are defined within the server scope:

- Are available for applications that are deployed on this server
- Override any resource factories that have the same JNDI name defined within the node and cell scopes

The resource factories that are defined within the node and cell scopes are available for this server to use, in addition to the resource factories that are defined within this server scope.

Application

Limits the visibility to the named application. Application scope resources can be viewed and edited from the console, but not created. You can additionally use the Rational Application Developer or the wsadmin tool to view or edit the application scope resource configuration. The resource factories that are defined within the application scope are available for this application to use only. The application scope overrides all other scopes.

You can view the application scope resources from the console by selecting **Applications** from the console navigation, and then navigating to the appropriate application. The application scope resources are unavailable from the Resources section of the console navigation.

You can configure namespace bindings and shared libraries under cell, node, and server scopes only. For WebSphere Application Server and Express, you can configure resources and the product variables under all four scopes.

Related tasks

“Specifying console preferences” on page 44

Use this topic to customize how much data displays on an administrative console panel.

Accessing help and product information from the administrative console

This topic describes how to use administrative console help and how to link to product documentation from the administrative console.

Before you begin

You must have a connection to the Internet to access information about WebSphere Application Server from the Welcome page of the administrative console.

About this task

All of the help panels that you can access from the administrative console, you can access from the WebSphere Application Server Information Center. This topic describes how to access the help panels, the information center, and other product documentation from the administrative console.

- Click **Welcome** on the administrative console navigation tree. In the workspace to the right of the navigation tree, click the link, which takes you to a page that has links to various documentation. The documentation that is linked includes the WebSphere Application Server Information Center, the WebSphere Application Server product information, and the WebSphere Application Server technical information on developerWorks®.
 - Access help in the following ways:
 - Click either of the following tabs of an online help page:
 - Click the **Help index** tab and select from the list of help panels to view administrative console help information.
 - Click the **Search** tab, provide search terms, and then click **Search**. Under Results, select a help panel that contains the search information.
 - In the help portal that is on the right side of the administrative console panel, do one or all of the following tasks:
 - Click a field label or a list marker in the administrative console panel for the help to display under Field help. Alternatively, place the cursor over the field label or the list marker for the corresponding help to display at the cursor.
- Note:** When you place the cursor over the field label or list marker, the help might be truncated in a Firefox browser. Click the field label or list marker so that the full help displays under Field help.
- Click the link under Page help to access the help panel for the administrative console panel. The help panel is the same help panel that displays when you click the **?** icon.
 - If Command assistance is listed, click the link under Command assistance to view wsadmin scripting commands for the last action run for this console panel.

What to do next

You can continue to access help information from the administrative console. Alternatively, you can access the help information from the WebSphere Application Server Information Center.

You can continue to access the WebSphere Application Server Information Center, the WebSphere Application Server product information, and the WebSphere Application Server technical information on developerWorks from the administrative console. Alternatively you can access the information from the IBM Web site.

Accessing command assistance from the administrative console

Using command assistance, you can view wsadmin scripting commands in the Jython language for the last action run in the administrative console. This topic discusses how to access command assistance from the administrative console.

Before you begin

You must have WebSphere Application Server and the administrative console running to access command assistance.

About this task

Use command assistance to see wsadmin scripting commands that correspond to actions in the administrative console. Seeing these commands might help you develop the commands necessary to administer WebSphere Application Server from the wsadmin utility.

If a command assistance link is listed in the help portlet, **wsadmin** commands exist for the last console action that you completed, and command assistance is available for that action.

When command assistance is unavailable in the help portlet: Some console actions do not have **wsadmin** commands directly associated with them. When the help portlet on the right side of the administrative console panel does not have a command assistance link in it, no command assistance data is available for the last console action.

1. Click the link under **Command assistance** to view wsadmin scripting commands for the last action run for this administrative console panel.

After the Command assistance window opens, it refreshes automatically when new command assistance data is available.

Examples of actions include a click on a button or a click on a link in the navigation bar, a collection panel, or a detail panel. The editing of forms is not a user action.

The wsadmin scripting commands display in the Jython language in a secondary window that you can view by clicking on the Command assistance link in the help portlet.

If you perform an administrative console action after you launch the Command assistance window, whether or not the scripting commands display in the window depends on whether your browser supports Java scripts. If your browser supports Java scripts, the Command assistance window automatically refreshes the command list to reflect the most recent console action. If the browser does not support Java scripts, click the link again under **Command assistance** in the help portal to refresh the command list.

2. To view the description of a specific **wsadmin** command, place your cursor over the command. Hover text is displayed.
3. Optionally, log the command assistance data to a file by selecting the **Log command assistance commands** setting on the Preferences page of the administrative console.

A timestamp and the breadcrumb trail of the panel that produced the command assistance data are provided with the wsadmin data.

4. Optionally, allow command assistance to emit Java Management Extensions (JMX) notifications by selecting the **Enable command assistance notifications** setting on the Preferences page of the administrative console.

Enablement of the notifications allows integration with product tools such as the Rational Application Developer Jython editor to assist you in writing scripts.

The notification type is `websphere.command.assistance.jython.user_name` where *user_name* is the name of the administrative console user.

Results

You have viewed `wsadmin` scripting commands from the administrative console, optionally logged the commands to a file, and optionally allowed command assistance to emit JMX notifications.

What to do next

You can continue your administration of the administrative console.

Administrative console actions with command assistance:

Using command assistance, you can view `wsadmin` scripting commands in the Jython language for the last action that runs in the administrative console. This topic lists the administrative console actions that have **`wsadmin`** commands available in the command assistance option of the Help portlet.

The table lists the components and the actions in the administrative console that have command assistance for a particular component. The administrative console can be an application server administrative console or an administrative agent administrative console. Listed actions might apply to one or both administrative consoles.

Component	Action
Applications	<ul style="list-style-type: none"> • List the applications. • Install the application. • Update the complete application. • Start the application. • Stop the application. • Edit the last participant support configuration. • Edit the deployment target mapping. • List Structured Query Language in Java (SQLJ) profiles. • Customize and bind SQLJ profiles. • List pureQuery bind files (*). • Bind pureQuery bind files (*). <p>(*) Command assistance in the administrative console produces <code>wsadmin</code> commands for SQLJ. These commands work with pureQuery bind files. Command assistance does not produce separate commands for the pureQuery bind files. However, in the <code>wsadmin</code> environment separate commands exists for you to use when working with pureQuery bind files.</p>
Web servers	<ul style="list-style-type: none"> • Create a Web server. • Delete a Web server. • Generate a plug-in configuration. • Propagate the plug-in configuration. • Propagate the key ring for the plug-in configuration. • Start the Web server. • Stop the Web server. • Terminate the Web server.

Component	Action
Channel framework	<ul style="list-style-type: none"> • List the SSL repertoires. • List the Transmission Control Protocol (TCP) endpoints. • List the TCP thread pools. • Delete a chain. • Get the TCP endpoint. • Create a TCP endpoint. • Create a chain.
Servers	<ul style="list-style-type: none"> • Create an application server. • Delete an application server. • Modify an application server. • List application servers. • Remove application servers. • Start an application server. • Stop an application server. • Create a custom property for an application server. The action is supported for an application server, but not its template. • Modify an existing custom property for an application server. • List custom properties for an application server. The action is supported for an application server, but not its template. • Remove custom properties for an application server. • Create a custom service for an application server. The action is supported for an application server, but not its template. • Modify an existing custom service for an application server. • List custom services for an application server. The action is supported for an application server, but not its template. • Remove custom services for an application server. • Modify application server components. • Modify the Object Request Broker (ORB) service. • Modify the ORB.thread.pool properties for an ORB service. • Modify the thread pool detail for an ORB service.

Component	Action
Servers (continued)	<ul style="list-style-type: none"> • Create a new class loader. The action is supported for an application server, but not its template. • Modify an existing class loader. • List class loaders. The action is supported for an application server, but not its template. • Remove class loaders. • Modify a process definition detail. • Create a new environment entry for an application server. The action is supported for an application server, but not its template. • Modify an existing environment entry for an application server. • List environment entries for an application server. The action is supported for an application server, but not its template. • Remove environment entries for an application server. • Modify Java virtual machine (JVM) configuration properties. • Modify JVM runtime properties.
Servers (continued)	<ul style="list-style-type: none"> • Modify process execution properties. • Modify process logs configuration properties. • Modify process logs runtime properties. • Create a new port property for the application server. • Modify an existing port property for the application server. • List ports for the application server. • Remove ports from the application server. • Modify session management properties. • Modify cookie properties. • Modify distributed environment settings. • Modify custom tuning parameters. • Modify custom settings for custom tuning parameters. • Modify database settings. • Create a new thread pool. The action is supported for an application server, but not its template. • Modify an existing thread pool. • List thread pools. The action is supported for an application server, but not its template. • Remove thread pools. • Create an application server template. • Delete an application server template. • List the application server templates.

Component	Action
Servers (continued)	<ul style="list-style-type: none"> • Create a generic server. • Delete a generic server. • Create a new environment entry for a generic server. The action is supported for a generic server, but not its template. • List environment entries for a generic server. The action is supported for a generic server, but not its template. • Modify the Enterprise JavaBeans (EJB) container settings. • Modify the EJB cache settings. • Modify the EJB timer service settings. • Modify the application profiling service. • Modify the internationalization service. • Modify the compensation service. • Modify the object pool service. • Modify the startup beans service. • Modify the ActivitySession service. • Modify the work area service. • Modify the core group service. • Create a work area partition. • Modify a work area partition. • Delete a work area partition.

Component	Action
Servers (continued)	<ul style="list-style-type: none"> • Modify Web container properties. • Create a custom property. The action is supported for a server, but not its template. • Modify an existing custom property. • List custom properties. The action is supported for a server, but not its template. • Remove custom properties. • Create a new Web container transport chain. • Modify an existing Web container transport chain. • Remove Web container transport chains. • Create an external cache group. • Edit an external cache group. • Edit a denial of service protection. • Modify the default Java persistence application programming interface (API) settings. • View product information. • View installed components. • View installed extensions. • Modify administrative services. • Modify a repository service. • Modify a generic server. • Create a generic server endpoint. • Modify a generic server endpoint. • Remove a generic server endpoint.
Session Initiation Protocol (SIP) container	<ul style="list-style-type: none"> • Modify the Session Initiation Protocol (SIP) container. • Modify the SIP stack. • Modify SIP timers. • Modify the SIP digest authentication. • List SIP application routers. • Modify a SIP application router.

Component	Action
Environment	<ul style="list-style-type: none"> • Create a name space binding for different types. • Edit an EJB name space binding. • Edit a name space binding of indirect lookup. • Edit a name space binding of other context property. • Edit a string name space binding. • Create bootstrap properties. • Edit bootstrap properties. • Create a group of Universal Resource Identifier (URI) patterns. • Edit a group of URI patterns. • Create a shared library. • Edit a shared library. • Modify a shared library. • List shared libraries. • Remove shared libraries. • Create a virtual host. • Edit a virtual host. • Modify a virtual host.
Environment (continued)	<ul style="list-style-type: none"> • List virtual hosts. • Remove virtual hosts. • Create a host alias for a virtual host. • Edit a host alias for a virtual host. • Modify host aliases for a virtual host. • List host aliases for a virtual host. • Remove host aliases for a virtual host. • Create a multi-purpose internet mail extensions (MIME) type. • Edit a MIME type. • Modify a MIME type. • List MIME types. • Remove MIME types. • Create a WebSphere variable. • Edit a WebSphere variable. • Modify a WebSphere variable. • List WebSphere variables. • Remove WebSphere variables.

Component	Action
Resources	<ul style="list-style-type: none"> • Create a Java Database Connector (JDBC) provider. • List the JDBC providers. • Modify a JDBC provider. • Delete a JDBC provider. • List the resource adapters. • Install a resource adapter. • Copy a resource adapter. • Upgrade a resource adapter. • Modify a resource adapter. • Delete a resource adapter. • Modify the advanced resource adapter properties of a resource adapter. • Create a custom property. • Modify a custom property. • Delete a custom property. • List the data sources. • Create a data source. • Modify a data source. • Remove a data source. • Modify the connection pool properties of a data source. • Modify the advanced connection pool properties of a data source. • Modify the Websphere Application Server data source properties of a data source.

Component	Action
Resources (continued)	<ul style="list-style-type: none"> • List the WebSphere Application Server Version 4 data sources. • Create a WebSphere Application Server Version 4 data source. • Modify a WebSphere Application Server Version 4 data source. • Delete a Websphere Application Server Version 4 data source. • Modify the connection pool properties of a Websphere Application Server Version 4 data source. • List the Java 2 Connector (J2C) connection factories. • Create a J2C connection factory. • Modify a J2C connection factory. • Delete a J2C connection factory. • Modify the connection pool properties of a J2C connection factory. • Modify the advanced connection pool properties of a J2C connection factory. • Modify the advanced connection factory properties. • List the J2C activation specifications. • Create a J2C activation specification. • Modify a J2C activation specification. • Delete a J2C activation specification. • List the J2C administered objects. • Create a J2C administered object. • Modify a J2C administered object. • Delete a J2C administered object.

Component	Action
Resources (continued)	<ul style="list-style-type: none"> • List the schedulers. • Create a scheduler. • Modify a scheduler. • Delete a scheduler. • Create the tables of a scheduler. • Verify the tables of a scheduler. • Drop the tables of a scheduler. • List the object pool managers. • Create an object pool manager. • Modify an object pool manager. • Delete an object pool manager. • Create a custom object pool. • Modify a custom object pool. • Delete a custom object pool. • List the work managers. • Create a work manager. • Modify a work manager. • Delete a work manager. • List the timer managers. • Create a timer manager. • Modify a timer manager. • Delete a timer manager. • Create a mail provider. • Modify a mail provider. • Create a mail session. • Edit a mail session. • Create a protocol provider. • Modify a protocol provider.
Resources (continued)	<ul style="list-style-type: none"> • Create a referenceable. • Modify a referenceable. • Create resource environment entries. • Edit resource environment entries. • Create a resource environment provider. • Edit a resource environment provider. • Create a URL. • Modify a URL. • Create a URL provider. • Modify a URL provider. • Modify an object cache instance. • Modify a servlet cache instance.

Component	Action
Transaction services	<ul style="list-style-type: none"> • List the manual transactions. • List the retry transactions. • List the heuristic transactions. • List the imported prepared transactions. • Set the total transaction lifetime timeout. • Set the asynchronous response timeout. • Enable file locking. • Enable transaction coordination authorization. • Set the client inactivity timeout. • Set the maximum transaction timeout.
Security	<ul style="list-style-type: none"> • Enable security. • Validate Lightweight Directory Access Protocol (LDAP) connections. • List SSL configurations. • Get the SSL configuration. • Create an SSL configuration. • Modify the SSL configuration. • Delete an SSL configuration. • List the SSL ciphers. • List the SSL configuration groups. • Create an SSL configuration group. • Delete an SSL configuration group. • Modify an SSL configuration group. • Get the inherited SSL configuration. • List dynamic outbound endpoint SSL configurations. • Create a dynamic outbound endpoint SSL configuration. • Delete a dynamic outbound endpoint SSL configuration. • List the key sets. • Generate a key for a key set. • Create a key set. • Delete a key set. • List the key set groups. • Create a key set group. • Delete a key set group. • Generate keys for the key set group.

Component	Action
Security (continued)	<ul style="list-style-type: none"> • List the keystores. • Create a keystore. • Modify a keystore. • Delete a keystore. • Change the keystore password. • Exchange signers. • List the key managers. • Create a key manager. • Delete a key manager. • List the key file aliases. • Create the key reference. • List the trust managers. • Create a trust manager. • Delete a trust manager. • List the certificate authority clients. • Create a certificate authority client. • Modify a certificate authority client. • Delete a certificate authority client. • List the personal certificates. • Get the attributes of a personal certificate. • Get a certificate chain. • Receive a personal certificate. • Create a self-signed certificate. • Create a chained certificate. • Create a certificate authority signed certificate.

Component	Action
Security (continued)	<ul style="list-style-type: none"> • Renew a personal certificate. • Revoke a personal certificate. • Replace a personal certificate. • Extract a personal certificate. • Import a personal certificate. • Export a personal certificate. • Delete a personal certificate. • Add a signer certificate. • Extract a signer certificate. • Retrieve signer information from a port. • Retrieve a signer certificate from a port. • Get the properties of a signer certificate. • Delete a signer certificate. • List the signer certificates. • Create a certificate request. • Get a certificate request. • List the certificate requests. • Delete a certificate request. • Extract a certificate request. • Query a certificate request. • List the notifiers. • Create a notifier. • Delete a notifier. • Start the certificate expiration monitor. • Validate the administrative name. • Add a base entry to the realm. • Modify the base entry details. • Configure a new LDAP repository. • Modify an existing LDAP repository configuration. • Delete an existing LDAP repository configuration.

Component	Action
Security (continued)	<ul style="list-style-type: none"> • View performance data for the LDAP repository under an LDAP configuration. • Modify the LDAP performance data. • View the LDAP entity types under the LDAP configuration. • Modify the existing LDAP entity types. • View the group attribute definition under the LDAP configuration. • Modify the group attribute definition under the LDAP configuration. • View the member attributes under the LDAP group attribute definitions. • Configure the member attribute details under the LDAP group attribute definitions. • Delete an existing member attribute detail. • View the dynamic member attributes under the LDAP group attribute definitions. • Configure the dynamic member attributes under the LDAP group attribute definitions. • Delete an existing dynamic member attribute detail. • View the list of repositories to manage. • Configure the federated repositories to use a built-in repository. • Remove the built-in repository from the federated repository configuration. • View the federated repository property extension. • Configure the federated repository property extension. • View the federated repository entry mapping repository. • Configure the federated repository entry mapping repository. • View the federated repository supported entity types list. • View the details of a supported entity type. • Modify an existing supported entity type. • View the authentication mechanism and expiration policy for the federated repository user identity. • Get an audit policy. • Modify an audit policy. • List the audit event type filters. • Get an audit event type filter. • Create an audit event type filter. • Modify an audit event type filter.

Component	Action
Security (continued)	<ul style="list-style-type: none"> • Delete an audit event type filter. • List the audit service providers. • Get an audit service provider. • Create an audit service provider. • Modify an audit service provider. • Delete an audit service provider. • List the audit factories. • Get an audit factory. • Create an audit factory. • Modify an audit factory. • Delete an audit factory. • List the audit encryption keystores. • Get an audit encryption keystore. • Create an audit encryption keystore. • Modify an audit encryption keystore. • Delete an audit encryption keystore. • Get an audit encryption configuration. • Create an audit encryption configuration. • Modify an audit encryption configuration. • Delete an audit encryption configuration. • Get an audit signing configuration. • Create an audit signing configuration. • Modify an audit signing configuration. • Delete an audit signing configuration. • List the audit notification monitors. • Create an audit notification monitor. • Modify an audit notification monitor, • List the audit notifications. • Create an audit notification • Modify an audit notification. • Delete an audit notification.

Component	Action
Security (continued)	<ul style="list-style-type: none"> • List the active security settings. • Set the active administrative security settings. • List the active Rivest Shamir Adleman (RSA) token authorization settings. • Set the active Rivest Shamir Adleman (RSA) token authorization settings. • List all authorization groups. • Create a new authorization group. • Delete an authorization group. • Edit an authorization group. • Get user realm information. • Configure a local OS user realm. • Configure an LDAP user realm. • Configure a custom user realm. • Unconfigure the user realm. • List trusted realms. • Add trusted realms. • Remove trusted realms. • Unconfigure the trusted realm. • Get external authorization provider information. • Set external authorization provider information. • Unconfigure external authorization provider. • List trust association interceptors. • Create a trust association interceptor. • Modify a trust association interceptor. • Delete a trust association interceptor. • Unconfigure the trust association. • Get Common Secure Interoperability (CSI) inbound information. • Set CSI inbound information. • Unconfigure CSI inbound information. • Get CSI outbound information. • Set CSI outbound information. • Unconfigure CSI outbound information.

Component	Action
Security (continued)	<ul style="list-style-type: none"> • List Java Authentication and Authorization Service (JAAS) login configurations. • Create JAAS login configurations. • Modify JAAS login configurations. • Delete JAAS login configurations. • Unconfigure JAAS login information. • Configure a JAAS login module. • Delete a JAAS login module. • List JAAS authorization data entries. • Create a JAAS authorization data entry. • Modify the JAAS authorization data entry. • Delete a JAAS authorization data entry. • Get Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) information. • Configure SPNEGO information. • List SPNEGO filters. • Create a SPNEGO filter. • Modify the SPNEGO filter. • Delete a SPNEGO filter. • Create a Kerberos authentication mechanism. • Modify the Kerberos authentication mechanism. • List security domains. • Create a security domain. • Copy the security domain. • Modify the security domain. • Delete the security domain. • Set active security settings. • Unset active security settings. • Set the Lightweight Third-Party Authentication (LTPA) timeout.

Component	Action
Service integration	<ul style="list-style-type: none"> • Create a bus. • Delete a bus. • Add a bus member. • Delete a bus member. • List the bus members. • Delete the messaging engine. • Create a queue. • Create a topic space. • Create an alias destination. • Delete an alias destination. • Create an MQ queue type destination. • Create a foreign destination. • Mediate a destination. • Unmediate a destination. • Delete a destination. • Create a mediation. • Delete a mediation. • Modify a mediation. • Delete a foreign bus. • Create a Java Message Service (JMS) activation specification. • Modify a JMS activation specification. • Create a JMS connection factory. • Modify a JMS connection factory. • Create a JMS queue connection factory. • Modify a JMS queue connection factory. • Create a JMS topic connection factory. • Modify a JMS topic connection factory.

Component	Action
Service integration (continued)	<ul style="list-style-type: none"> • Create a JMS queue. • Modify a JMS queue. • Create a JMS topic. • Modify a JMS topic. • Create a JMS provider. • Create a WebSphere MQ server. • Modify a WebSphere MQ server. • Modify a WebSphere MQ server bus member. • Add a permitted transport. • Add a user to a bus connector role. • Add a group to a bus connector role. • Add an inbound port to an inbound service. • Add an outbound port to an outbound service. • Connect an endpoint listener to a service integration bus. • Create an endpoint listener. • Create an inbound service. • Create an outbound service. • Delete an endpoint listener. • Delete an inbound service. • Delete an outbound service. • Disconnect an endpoint listener from a service integration bus. • Publish an inbound service to a Universal Description, Discovery, and Integration (UDDI) registry. • Refresh the Web Services Description Language (WSDL) definition for an inbound service. • Refresh the WSDL definition for an outbound service. • Remove an inbound port. • Remove an outbound port. • Set the default outbound port for an outbound service. • Remove an inbound service from a UDDI registry. • Create a WS-Notification (WSN) service.

Component	Action
Service integration (continued)	<ul style="list-style-type: none"> • Modify a bus. • Modify a foreign bus connection. • List foreign bus connections. • List bus messaging engines. • List messaging engines on a particular server. • List bus mediations. • Modify a messaging engine. • Create a service integration bus (SIB) link on a messaging engine. • Modify a foreign bus connection (an indirect link). • Modify a foreign bus connection (a SIB link). • List bus destinations. • List queue points (point-to-point messaging) for a destination. • List queue mediation points for a destination. • List topic mediation points for a destination. • Create a new context property. • Edit a bus. • Create a replication domain. • Edit a replication domain.
System administration	<ul style="list-style-type: none"> • Edit the file synchronization service. • Edit the file transfer service. • Modify a cell. • Modify the deployment manager. • Modify a node. • Modify a node agent. • Modify a Java Management Extensions (JMX) connector. • Modify extension MBean providers. • Modify node groups.
Web services	<ul style="list-style-type: none"> • List the service clients in a cell. • List the service clients in an application. • List the service providers in a cell. • List the service providers in an application. • Start the service provider listener. • Stop the service provider listener.

Component	Action
Web services policy sets	<ul style="list-style-type: none"> • List the policy sets in the system. • Get the attributes for a policy set. • Set the attributes of a specified policy set. • Create a new policy set. • Copy a policy set to create a new policy set. • Delete a policy set. • Update the attributes of a policy set. • Export a policy set from an archive for use in a client environment or a server. • Get the policy set attachments for a given resource. • List the policy set attachments for services providers. • List the policy set attachments for services clients in an application. • Create a new policy set attachment for a resource. • Delete a policy set attachment from a resource. • List the applications to which a given policy set is attached. • Delete all attachments for a policy set. • Transfer all attachments from one policy set to another.
Web services policies	<p>Note: Before you use the generated wsadmin command for policies, see the documentation for the PolicySetManagement command group for the AdminTask object.</p> <ul style="list-style-type: none"> • Create a policy. • Add a policy to a policy set. • Delete a policy from a policy set. • List the names of existing policies. • Get the attributes for a policy. • Update the configuration of a policy. • Get the value for a named policy attribute. • Set the value for a named policy attribute.
Web services bindings	<ul style="list-style-type: none"> • Get the binding configuration for a specified policy for a policy set attachment. • Set the binding for a policy set attachment. • Set and update the binding configuration for a specified policy for a policy set attachment.

Component	Action
Web services trust service	<ul style="list-style-type: none"> • List the local names of all the configured token providers. • Query the trust service for the local name of the default token provider. • Update configuration data for a token provider. • Delete custom properties from a token provider configuration. • Assign a token that is issued when requesting access to a specific end point. • List the assigned endpoints for a token provider. • Query the trust service for the token provider assigned to a specified endpoint. • Unassign an endpoint from its token provider. • Refresh trust service. • Query the trust service for a list of assigned endpoints.
Web services security distributed cache	<ul style="list-style-type: none"> • Get the Web services security distributed cache configuration. • Get the Web services security distributed cache configuration custom properties. • Get the defined cell level data sources. • Set the Web services security distributed cache configuration. • Set the Web services security distributed cache configuration custom properties.
Business-level applications	<ul style="list-style-type: none"> • List the assets. • Add an asset to the repository. • Export an asset. • Delete an asset . • List the business-level applications. • Create a new business-level application. • Add an asset to a business-level application. • Delete a business-level application. • Edit a business-level application. • Start a business-level application. • Stop a business-level application. • Edit a composition unit.
Performance Monitoring Infrastructure (PMI) and Request metrics	<ul style="list-style-type: none"> • List the PMI configuration. • Modify the PMI configuration. • Modify the PMI parameters at run time. • Edit the request metrics filter. • Edit the request metrics. • Create a request metrics filter value. • Edit a request metrics filter value. • Delete the request metrics filter.
Performance advisors	<ul style="list-style-type: none"> • Re-initialize the Runtime Performance Advisor tool. • Set the Runtime Performance Advisor tool attributes.

Component	Action
Portlets and portlet containers	<ul style="list-style-type: none"> • View the portlet deployment descriptor. • Modify portlet container settings. • Create a custom property. • Modify an existing custom property. • List custom properties. • Remove custom properties. • Enable PMI for portlets. • Enable request metrics for portlets.
DataPower®	<ul style="list-style-type: none"> • Add an appliance to the DataPower appliance manager. • Add a firmware version to the DataPower appliance manager. • Add a managed set to the DataPower appliance manager. • Copy a DataPower appliance manager managed domain version to a new managed set. • Copy a DataPower appliance manager managed settings version to a new managed set. • Export the configuration and versions of a DataPower appliance manager. • Get the IDs of all the DataPower appliance manager appliances. • Get the names of all the domains on a DataPower appliance. • Get the IDs of all the DataPower appliance manager firmware. • Get the IDs of all the DataPower appliance manager firmware versions. • Get the IDs of all the appliances in a DataPower appliance manager managed set. • Get the IDs of all the domains in a DataPower appliance manager managed set. • Get the IDs of all the domain versions in a DataPower appliance manager managed set. • Get the IDs of the managed sets using a firmware version. • Get the IDs of all the settings versions in a DataPower appliance manager managed set. • Get the IDs of all the DataPower appliance manager managed sets. • Get the IDs of all the DataPower appliance manager tasks. • Get the firmware that best matches the parameters. • Get a DataPower appliance manager appliance. • Get a DataPower appliance manager firmware. • Get a DataPower appliance manager firmware version. • Get a DataPower appliance manager managed domain.

Component	Action
DataPower (continued)	<ul style="list-style-type: none"> • Get a DataPower appliance manager managed domain version. • Get a DataPower appliance manager managed settings. • Get a DataPower appliance manager managed settings version. • Get a DataPower appliance manager managed set. • Get the properties of the DataPower appliance manager. • Get a DataPower appliance manager task. • Import the configuration and versions of the DataPower appliance manager. • Tell the DataPower appliance manager to add the appliance to a managed set and start managing the appliance. • Tell the DataPower appliance manager to add the domain to a managed set and start managing the appliance. • Purge a DataPower appliance manager task. • Remove an appliance from the DataPower appliance manager. • Remove a firmware version from the DataPower appliance manager. • Remove a managed domain version from the DataPower appliance manager. • Remove a managed settings version from the DataPower appliance manager. • Remove a managed set from the DataPower appliance manager. • Modify a DataPower appliance manager appliance. • Modify a DataPower appliance manager firmware version. • Modify a DataPower appliance manager managed domain. • Modify a DataPower appliance manager managed domain version. • Modify a DataPower appliance manager managed settings. • Modify a DataPower appliance manager managed settings version. • Modify a DataPower appliance manager managed set. • Modify the DataPower appliance manager.

Component	Action
DataPower (continued)	<ul style="list-style-type: none"> • Manually synchronize a DataPower appliance manager managed set. • Tell the DataPower appliance manager to remove the appliance from its managed set and stop managing the appliance. • Tell the DataPower appliance manager to remove the domain from its managed set and stop managing the appliance. • Get the status of the DataPower appliance manager. • Shut down the DataPower appliance manager.
Replication Domains	<ul style="list-style-type: none"> • List data replication domain members.
Users and groups	<ul style="list-style-type: none"> • Add a new user. • Modify an existing user. • List users. • Remove users. • Add a new group. • Modify an existing group. • List groups. • Remove groups. • List registry users. • List registry groups.

Administrative console: Resources for learning

Use the following links to find relevant supplemental information about the IBM WebSphere Application Server administrative console. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and IBM Redbooks® that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information:

Administration

- IBM WebSphere Application Server Redbooks
This site contains a listing of all WebSphere Application Server Redbooks.
- IBM WebSphere developerWorks
This site is the home of technical information for developers working with WebSphere products. You can download WebSphere software, take a fast path to developerWorks zones, such as VisualAge® Java or WebSphere Application Server, learn about WebSphere products through a newcomers page, tutorials, technology previews, training, and Redbooks, get answers to questions about WebSphere products, and join the WebSphere community, where you can keep up with the latest developments and technical papers.
- WebSphere Application Server Support page
Take advantage of the Web-based Support and Service resources from IBM to quickly find answers to your technical questions. You can easily access this extensive Web-based support through the IBM Software Support portal at Web address <http://www.ibm.com/software/support/> and search by product

category, or by product name. For example, if you are experiencing problems specific to WebSphere Application Server, click **WebSphere Application Server** in the product list. The WebSphere Application Server Support page appears.

Console identity

Use Console Identity to define a string (keyword, name, or phrase) to be displayed in the console banner. This string can be used to distinguish this console from other console instances.

To access this page, click **System administration** → **Console Identity** in the console navigation.

The console identity string is rendered differently in the banner and browser title bar.

- **Console identity in the banner**

The console identity string is displayed in the console banner after the greeting to the console user, separated by a dash. For example, if the identity string is set to Jupiter and the user is logged in as consoleadmin, the banner displays the following greeting.

Welcome consoleadmin - Jupiter

On the login page, the console identity is displayed without the greeting.

- **Console identity in the browser title bar**

The console identity string is displayed in the title bar after the console brand name, separated by a space. For example, if the identity string is set to Saturn, the title bar displays the following information.

Integrated Solutions Console Saturn

Note: After saving these changes, the console identity settings are applied across the console. For an administrative agent configuration, this means that the changes are applied to the administrative agent and all of its registered application servers, regardless of where the changes were actually saved.

Console identity

Specifies whether a custom string is used to identify this console.

none

Select this option if a custom string should not be displayed.

custom

Select this option to display a custom string in the banner and browser. When this option is selected, **Custom identity string** is enabled.

Custom identity string

Specifies the text string that you want to display in the banner and browser title bar.

Truncate string at (Characters)

Specifies how many characters of the string should be displayed. Input for this field must be an integer. Minimum value is 4, maximum value is 99. When the string is truncated, 3 of the characters are used for an ellipsis (...) to indicate that it has been truncated.

Console identity preview

Click **Preview** to see how the custom string will be displayed in the banner.

Changing the console session expiration

Run this JACL script to set how long Integrated Solutions Console can be used until the login session expires.

About this task

The following JACL script serves as an example of how to set the duration that an Integrated Solutions Console can be used until the login session expires. Other scripting types, such as JYTHON, could be used.

1. Copy the following script into a file.

```
set dep [$AdminConfig getid /Deployment:isclite/]
set appDep [$AdminConfig list ApplicationDeployment $dep]
set sesMgmt [$AdminConfig list SessionManager $appDep]

# check if existing sesMgmt there or not, if not then create a new one, if exist then modify it
if {$sesMgmt == ""} {
  # get applicationConfig to create new SessionManager
  set appConfig [$AdminConfig list ApplicationConfig $appDep]
  if {$appConfig == ""} {
    # create a new one
    set appConfig [$AdminConfig create ApplicationConfig $appDep {}]
    # then create a new SessionManager using new Application Config just created
    set sesMgmt [$AdminConfig create SessionManager $appConfig {}]
  } else {
    # create new SessionManager using the existing ApplicationConfig
    set sesMgmt [$AdminConfig create SessionManager $appConfig {}]
  }
}

# get tuningParams config id
set tuningParams [$AdminConfig showAttribute $sesMgmt tuningParams]
if {$tuningParams == ""} {
  # create a new tuningParams
  $AdminConfig create TuningParams $sesMgmt {{invalidationTimeout <timeout value>}}
} else {
  #modify the existing one
  $AdminConfig modify $tuningParams {{invalidationTimeout <timeout value>}}
}

# saving the configuration changes
$AdminConfig save
```

2. Change the *<timeout value>* on the two lines of this sample to the new session expiration value. This number specifies the number of minutes the console preserves the session during inactivity.
3. Save the file to any directory using, for example, the filename `timeout.jacl`.
4. Start the `wsadmin` scripting client from the `<WAS-install>/profiles/<profile_name>/bin` directory.
5. Issue the following command.

```
wsadmin -f <path to jacl file>/timeout.jacl
```

Changing the class loader order of the console module deployed in Integrated Solutions Console

Run this JACL script to change the class loader order of the console module deployed in the Integrated Solutions Console.

About this task

The following JAACL script serves as an example of how to change the class loader order of the console module deployed in the Integrated Solutions Console. Other scripting types, such as JYTHON, could be used.

1. Copy the following script into a file.

```
set app [$AdminConfig getid /Deployment:isclite/]
set webModules [$AdminConfig list WebModuleDeployment $app]

foreach webModule $webModules {
  set uri [$AdminConfig showAttribute $webModule uri]
  if {$uri == "<WAR_NAME>"} {
    #modify the classloader for <WAR_NAME>
    set cl [$AdminConfig list Classloader $webModule]
    # check if the classloader exist
    if {$cl == ""} {
      # create a new one with the appropriate mode
      $AdminConfig create Classloader $webModule {{mode <MODE>}}
    } else {
      # modify the existing one
      $AdminConfig modify $cl {{mode <MODE>}}
    }
  }
}

# save the configuration change
$AdminConfig save
```

2. Change the <WAR_NAME> on the two lines of this sample to the name of the console module file deployed in the Integrated Solutions Console which class loader order you want to change.
3. Change the <MODE> on the two lines of this sample to PARENT_LAST or PARENT_FIRST as required.
4. Save the file to any directory using, for example, the file name classloaderorder.jacl.
5. Start the wsadmin scripting client from the <WAS-install>/profiles/<profile_name>/bin directory.
6. Issue the following command.

```
wsadmin -f <path to jacl file>/classloader.jacl
```

Console identity string

You can define a console identity string to be displayed in the console banner and in the browser's title bar. This string can be fixed or you can use a variable that is resolved at run time with a Java system property or environment variable.

The console identity string is rendered differently in the banner and browser title bar.

- **Console identity in the banner**

The console identity string is displayed in the console banner after the greeting to the console user, separated by a dash. For example, if the identity string is set to Jupiter and the user is logged in as consoleadmin, the banner displays the following greeting.

Welcome consoleadmin - Jupiter

On the login page, the console identity is displayed without the greeting.

- **Console identity in the browser title bar**

The console identity string is displayed in the title bar after the console brand name, separated by a space. For example, if the identity string is set to Saturn, the title bar displays the following information.

Integrated Solutions Console Saturn

After installation, there are two ways provided to customize the identity string.

- Use Console Identity in the administrative console. To open, log in to the administrative console and click **System administration** → **Console Identity**.
- Set the value for the string in an XML file, consoleProperties.xml, which is located in *app_server_root/systemApps/isclite.ear/isclite.war/WEB-INF*. The schema definition for this file, ibm-portal-console-properties.xsd, is also located in the same directory. You should keep a backup copy of the most recent working consoleProperties.xml before making changes to this file.

The keys that are used to set the console identity string to be displayed in the banner and title bar are defined in an XML <console-property/> element. The required attributes **id** and **value** for this element determine the key name and value pairs. The following initial settings are provided in consoleProperties.xml.

(Some keys are split on multiple lines for printing purposes.)

Table 1. Console identity key values

Key (id)	Initial setting (value)	Description
ISC.CONSOLE.ID	ISC.CONSOLE.ID.CUSTOM	Specifies the ID attribute of another <console-property/> element that contains the actual console identity string
ISC.CONSOLE.ID.MAXLEN	27	Specifies the maximum length of the resolved console identity string. The string will be truncated to this length if necessary.
ISC.CONSOLE.ID.CUSTOM	<i>empty string</i>	A custom string to be displayed in the banner and title bar.
ISC.CONSOLE.ID.DEPLOYER. <i>unique_suffix</i>	<i>None</i>	Specifies a custom string that can be selected using the Console Identity application under System Administration. By providing multiple <console-property/> entries with the ISC.CONSOLE.ID.DEPLOYER. <i>unique_suffix</i> id attribute, you can provide multiple identity strings for console users to select from. Each entry of this type should have a distinct <i>unique_suffix</i> from the other entries. If multiple entries are provided with the same <i>unique_suffix</i> , then only the first value with this suffix is displayed in the selection list for the console user.

Note:

- In the following example, the deployer wants to create custom strings that indicate different departments served by their respective host consoles. Console users will be able to select which identity is displayed in each department's console interface. To achieve this goal, the deployer provides the following custom keys in the installation package. Out of the box, the blank value for ISC.CONSOLE.ID.CUSTOM is used.

```

<?xml version="1.0" encoding="UTF-8"?>
<consoleproperties:ibm-portal-consoleproperties xmlns:consoleproperties=
  "http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-consoleproperties.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-consoleproperties.xsd
      ibm-portal-consoleproperties.xsd">
  <consoleproperties:console-property id="ISC.CONSOLE.ID"
    value="ISC.CONSOLE.ID.CUSTOM"/>
  <consoleproperties:console-property id="ISC.CONSOLE.ID.MAXLEN" value="27"/>
  <consoleproperties:console-property id="ISC.CONSOLE.ID.CUSTOM" value=""/>
  <consoleproperties:console-property
    id="ISC.CONSOLE.ID.DEPLOYER.sales" value="Sales & Marketing"/>
  <consoleproperties:console-property
    id="ISC.CONSOLE.ID.DEPLOYER.finance" value="Finance"/>
  <consoleproperties:console-property
    id="ISC.CONSOLE.ID.DEPLOYER.research" value="Research & Development"/>
  <consoleproperties:console-property
    id="ISC.CONSOLE.ID.DEPLOYER.dist" value="Distribution"/>
</consoleproperties:ibm-portal-consoleproperties>

```

- The administrator wants to append information about the host console to the banner and title bar. This can be accomplished by updating the ISC.CONSOLE.ID.CUSTOM key to include the PROCESSOR_ARCHITECTURE environment variable (which works on Windows machines) along with the **os.name** Java system property. The other keys are left unchanged.

```

<?xml version="1.0" encoding="UTF-8"?>
<consoleproperties:ibm-portal-consoleproperties
  xmlns:consoleproperties=
    "http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-consoleproperties.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-consoleproperties.xsd
      ibm-portal-consoleproperties.xsd">
  <consoleproperties:console-property id="ISC.CONSOLE.ID"
    value="ISC.CONSOLE.ID.CUSTOM"/>
  <consoleproperties:console-property id="ISC.CONSOLE.ID.MAXLEN" value="27"/>
  <consoleproperties:console-property id="ISC.CONSOLE.ID.CUSTOM"
    value="Platform: %PROCESSOR_ARCHITECTURE% OS: %os.name%"/>
</consoleproperties:ibm-portal-consoleproperties>

```

Since the ISC.CONSOLE.ID.MAXLEN key indicates a maximum display of 27 characters, the resulting string, which is resolved at run time, is rendered in the banner as follows for a Windows 2003 Server environment.

Platform: x86 OS: Window...

Note: After the console identity string is saved, the settings are applied across the console. For an administrative agent configuration, this means that the changes are applied to the administrative agent and all of its registered application servers, regardless of where the changes were actually saved.

Developing console modules

Console modules are Web applications that are accessed from Integrated Solutions Console. Console modules provide the business logic and transaction processes that enable administration functions.

About this task

The following skills are essential for developing and testing console modules.

- Java Platform, Enterprise Edition (Java EE)
- XML
- Portlet development using the Java Portlet Specification (JSR 168)

The console also supports existing modules that have been developed using Struts and Tiles APIs. This is only for legacy support. New console modules are supported only if they are developed using standard portlet APIs.

- Review the console module samples. The sample console modules provide examples of portal application archives and how to use the APIs and other features.
- Set up the development environment. You need an integrated development environment (IDE) that supports Java development to quickly develop, test, and deploy your portlet applications.
- Develop your first console module. This topic is for developers new to console module development. It is assumed that you already have an IDE prepared. The topics in this section take you through the process of creating a simple console module. The console module in this exercise consists of a single portlet which is deployed to a single page as a member of the sample console modules provided by IBM. Be sure you have successfully deployed the sample console modules before starting.
- Adding advanced API features. The class files for console modules are developed using the portlet API of the Java Portlet Specification (JSR 168). Integrated Solutions Console includes additional APIs for launching pages, passing properties to other console modules, and launching Eclipse-based help. This topic provides information about the APIs available to console modules.

Overview of Integrated Solutions Console

Integrated Solutions Console offers several advantages for implementing administration functions, as described in this topic.

Integrated Solutions Console provides a single, common interface for system administration. It provides the main platform on which IBM and non-IBM products can build administrative user interfaces as individual plug-ins to a common console framework. Standardizing product administration functions to run on the Integrated Solutions Console platform gives them a more common look and feel and a more consistent behavior, thereby reducing the learning curve and adoption as new management components are introduced. Administrators can interact with multiple IBM and non-IBM products from a single browser-based console.

Consistency across administrative interfaces

Integrated Solutions Console provides a common appearance (for example, theme, layout and banner) and behavior (for example, navigation and authentication) to enable consistent user interaction for administering software products.

A standards-based architecture

Integrated Solutions Console provides a standards-based architecture for Web administration. Each Integrated Solutions Console module consists of one or more Web applications that have access to services within the Java Platform, Enterprise Edition (Java EE) environment provided by WebSphere Application Server. The help interface is implemented using the Eclipse open standard. Console modules are developed using the Java Portlet Specification.

Easy deployment of product administration consoles

The Integrated Solutions Console framework provides an XML-based interface for deploying console modules to a console installation. XML descriptors provide the information needed to deploy the portlet,

resources, and setup the page layout and navigation in the console. A console module can be easily removed without impact to the remaining console modules.

Accelerated development of solutions

Using Integrated Solutions Console enables products to reduce the time required to develop solutions that require administration functions. The standards-based architecture, common framework, and support for existing investment help reduce the time required to implement solutions.

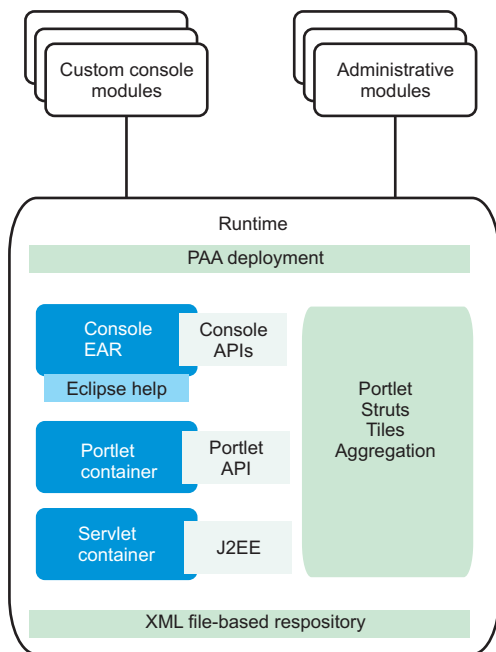
Improved administration efficiency

Customers invest significant resources in training administrative personnel. Providing a common interface across IBM products enables customers to reduce training time and expense.

Integrated Solutions Console components

This topic describes the components of the administrative console.

This figure illustrates the Integrated Solutions Console platform.



The following components and features support console modules running in Integrated Solutions Console.

- A common console
Provides a common appearance and navigation behavior for console modules. For example, each portlet is rendered using a common skin, rendering the same title bar icons for actions such as editing preferences, maximizing, or requesting help for the portlet.
- Built-in administration console modules
Provide security, log and trace, and other functions that apply to the entire the Integrated Solutions Console installation.
- A portlet container
Supports portlets developed using the Java Portlet Specification. The current specification as of this writing is JSR 168.
- PAA deployment

XML descriptors are provided for the developer to specify pages, navigation nodes, and access control for the console module, which are created at deployment. The descriptor can also register the following information about the console module.

- Custom roles for the application in addition to the roles provided by the application server
 - An about page with links to useful information and resources for a product
 - A pointer to another console module that is the lead application for the product
 - Prerequisite applications and console modules that should be already deployed
 - Portlets that are already deployed on the server that should be reused for the console module
 - Multiple entities of a single portlet definition that can be used on different pages
 - External URLs that can be launched from the console navigation
 - Row and column layout of a page
 - Windows that contain portlets on the page
 - Empty windows on a page that are filled at runtime depending on the user's navigation selection
 - The order and nesting of navigation nodes used by the console module
 - Parent navigation nodes under which the navigation tree for the current console module are placed
 - Navigation nodes from other products that should be included under the current console module's navigation
- An application server
Provides a complete Java runtime environment for console modules.
 - APIs for console modules
Supports the following actions:
 - Passing properties between portlets
 - Launching a page from a portlet
 - Invoking Eclipse-based help
 - Eclipse Help
Eclipse Help is provided as a Web application running on the same server as Integrated Solutions Console. Eclipse Help serves help documents for a single module as well as for the entire console.

What is new in Integrated Solutions Console

Learn about changes to the administrative console since the last release of WebSphere Application Server.

Customizing the banner

You can change the console banner to display custom text in the greeting. This could be useful, for example, in environments where you have multiple application servers and you want to make sure console users can determine which server they are administering. The same text is also displayed in the title bar of the browser window.

Administrative agent console

In environments that use the administrative agent to administer multiple application server nodes, you have a choice to select when logging in to the console. You can choose to log into the administrative agent or one of its registered profiles.

Example: Console module samples

The sample console modules provide examples of portal application archives and how to use the APIs and other features. These samples are available from the *Samples for WebSphere Application Server* page at the following location

<http://www.ibm.com/developerworks/websphere/library/samples/index.html>

These samples help understand the structure of a console module WAR package and the portal application descriptors that distinguish the application as a console module. After downloading the samples, follow the instructions in “Deploying a console module” on page 89. The following table lists the samples that are ready for deployment.

Sample	Description
Page layout (pagelayout.war)	Demonstrates how to specify the layout of console modules on a page using the elements of the portal topology descriptor. This sample shows how the lead application of a product provides an about page, which contains helpful information and resources about the product. The about page can be accessed from the console Welcome page. This module also demonstrates how to reuse a portlet from another module (Tree merge).
Tree merge (TreeMerge.war)	Demonstrates how a subordinate application deploys into the lead application for a product. The <PAA-ref/> element indicates the page layout sample as the lead application. Also, the <requires/> element indicates the page layout sample as a prerequisite that must already be deployed. The <parent-tree/> element merges the navigation nodes for the tree merge sample into the page layout navigation tree.
Portlet context (PortletContext.war)	Demonstrates how to use the Dynamic UI Manager APIs to launch a page and how to pass properties to other console modules. The module receiving these properties can be on the same page or on another page.
Modes (Modes.war)	Demonstrates how to implement the Edit and Help modes in a module and how to invoke Eclipse-based help.
Basic module (Basicmodule.war)	Demonstrates how to implement basic module components.

Setting up the development environment

Set up your integrated development environment (IDE) for developing and debugging console modules.

You can use the Integrated Solutions Console Visual Designer feature to create the portal topology and security deployment descriptors needed to enable portlet applications to run in the administrative console environment. Visual Designer is available for download at <http://www.ibm.com/support/docview.wss?rs=180&uid=swg24011666>. This feature is supported using one of the following IDEs.

- IBM Application Server Toolkit, Version 6.1.1
- IBM Rational Application Developer, Version 7.0

After you download Visual Designer, you can install it to your development environment like any other feature. Refer to the documentation for your IDE for more information.

As you test your console modules, you might discover that changes that you make to the JSP files are not rendered when you log in to the application server. If so, you probably need to check the **reloadingEnabled** attribute in the extended deployment descriptor for the module (ibm-web-ext.xmi) Make sure this value is set to true to enable the JSP file to reload each time you deploy a new change. See the “Hot deployment and dynamic reloading” topic in the *Developing and deploying applications* PDF for more information.

Developing your first console module

Follow these steps if you are new to console module development.

Before you begin

You should have an integrated development environment (IDE) that supports Java programming setup. See “Setting up the development environment” on page 83 for more information.

The topics in this section take you through the process of creating a simple console module. The sample console module consists of a single portlet which is deployed to a single page as part of the set of sample console modules provided by IBM. Be sure you have successfully deployed the sample console modules before starting.

- “Developing the portlet”
- “Creating the descriptors for the console module” on page 85
- “Packaging a console module” on page 89
- “Deploying a console module” on page 89
- “Testing a console module” on page 91
- “Removing a console module” on page 91

Developing the portlet

This topic is intended for developers who are unfamiliar with portlet development. A simple Java class and JSP are provided.

Developing the source Java class file

The following example shows the Java code for a portlet in its simplest form.

```
package com.ibm.isclite.samples.basicmodule;

import java.io.*;
import javax.portlet.*;

public class BasicModule extends GenericPortlet {

    public void doView(RenderRequest request, RenderResponse response)
        throws PortletException, IOException {
        // Set the MIME type for the render response
        response.setContentType("text/html");

        // Invoke the JSP to render
        PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher("/jsp/basicView.jsp");
        rd.include(request, response);
    }
}
```

The portlet code must extend the `GenericPortlet` class and output for the response in the `doView()` method. Portlets are rendered in different modes. The initial mode when a portlet is called to render is the view mode. The response output is provided by a JSP, which provides markup that can be aggregated into a larger HTML page. The package name in this example is consistent with the console module samples.

Developing the JSP

The following shows a portlet JSP for view mode in its simplest form.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" session="false" buffer="none"%>
<%@ page import="javax.portlet.*" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portletAPI" %>

<portletAPI:defineObjects />

<%
```



```

    PortletPreferences prefs = renderRequest.getPreferences();
    String URL = prefs.getValue("website","");
%>

<p><a name="<portletAPI:namespace/>basicAnchor">Basic contents</a></p>

<p>
  <a href="<%=URL%>" target="_blank">
    <img src='<%=renderResponse.encodeURL(renderRequest.getContextPath() + "/images/logo.gif")%>'
      alt="logo" />
    </a>
  </p>
<p><em>Company logo with link.</em></p>

```

The taglib directive specifies the portlet tags from the Java Portlet Specification. The following tags are used in this example.

namespace

Uniquely qualifies named HTML tags or JavaScript functions to prevent clashes with other portlets on the page using the same name.

defineObjects

Makes the RenderRequest, RenderResponse, and PortletConfig objects available to the JSP. This tag is used in this example so that the image can be rendered using the encodeURL() method of the render response, and to allow the JSP to read preferences in the portlet descriptor.

Creating the descriptors for the console module

Before you begin

The following descriptors are required for the console module.

- Web application descriptor (web.xml). This descriptor is required and described by the Java Servlet Specification.
- Portlet application descriptor (portlet.xml). This descriptor is required and described by the Java Portlet Specification. Refer to the console module samples for examples of how to create the portlet.xml.
- Portal topology descriptor (ibm-portal-topology.xml). This descriptor is unique to console modules and defines the portlets and resources in the module, how it is laid out on the page, and how it is accessed from the navigation.
- Portal security descriptor (ibm-portal-security.xml). This descriptor is unique to console modules and defines roles and access to navigation nodes and portlets.

About this task

The topology and security descriptors are the main component of console module packages that distinguish them from other portal application WAR files. The console module samples include these descriptors and demonstrate their features. The sample descriptors provide inline comments that explain the elements and how they are used.

For your first console module, you should use the Visual Designer plugin to the Application Server Toolkit to develop the descriptors. If you prefer to create them manually to become familiar with the structure of the XML, the steps in this topic show how to create simple descriptors that can be used to deploy an application into the samples provided by IBM. After following these steps, you should refer to the samples and the reference topics for the console module schemas for more complete information.

Portal topology descriptor:

About this task

To follow these steps, open a file with the name ibm-portal-topology.xml in a text or XML editor and save each change at the end of each step.

1. Copy the following elements into `ibm-portal-topology.xml`. Some information is split on multiple lines for printing purposes.

```
<?xml version="1.0" encoding="UTF-8"?>

<ibm-portal-topology
  xmlns="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-topology.xsd"
  xmlns:base="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-base.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-topology.xsd
ibm-portal-topology.xsd">

  <application-definition appID="com.ibm.isclite.samples.basicModule" version="6.1">

    </application-definition>
    <PAA-ref>com.ibm.isclite.samples.PageLayout</PAA-ref>

  </ibm-portal-topology>
```

- The elements in this step provide the top level content of the topology descriptor.
- Most of the content for this descriptor will be placed inside the `<application-definition/>` element.
- The value of the **appID** attribute is consistent with the naming convention used by the sample console modules. The **version** attribute matches the version used by the sample console modules.
- The `<PAA-ref/>` element specifies the **appID** of the page layout sample, which is the lead application to which this application will be deployed.
- The following convention is used for unique name for all of the elements in the descriptor.

namespace + element_type + identifier

2. Create a title for your application. The title is the first element in the application definition element.

```
<title>
  <base:nls-string lang="en">My basic module</base:nls-string>
</title>
```

3. Place the component tree after the `<requires/>` element as shown.

```
<component-tree uniqueName="com.ibm.isclite.samples.basicModule.appTree">
  <portlet-definition uniqueName="com.ibm.isclite.samples.basicModule.portletDefinition.A">
    <title>
      <base:nls-string lang="en">My basic module</base:nls-string>
    </title>
    <resource-link name="basicModule"
      portletApplication="com.ibm.isclite.samples.basicmodule.BasicModule.01a"
      type="static"/>
  </portlet-definition>
  <portlet-entity uniqueName="com.ibm.isclite.samples.basicModule.portletEntity.A"
    portletDefinitionRef="com.ibm.isclite.samples.basicModule.portletDefinition.A">
    <title>
      <base:nls-string lang="en">My basic module</base:nls-string>
    </title>
  </portlet-entity>
</component-tree>
```

- The **name** attribute of the `<resource-link/>` element must specify the `<portlet-name/>` from the portlet deployment descriptor.
- The **portletApplication** attribute of the `<resource-link/>` element must specify the **id** attribute of the `<portlet-app/>` element from the portlet deployment descriptor.
- The **portletDefinitionRef** attribute of the `<portlet-entity/>` element references the unique name of the `<portlet-definition/>`. Subsequent elements in this descriptor use the unique name of the `<portlet-entity/>` to invoke the portlet. Multiple portlet entities can point to the same portlet definition. For the purpose of this exercise, a single portlet definition is provided with a single portlet entity.
- The portlet title that is rendered in this example is derived from the title in the portlet entity. If that value is not provided, the portlet title is rendered by the title provided by the `<window/>` element in

the layout tree. If that value is also not provided, the title is obtained from the <portlet-name/> element in the portlet.xml for the portlet indicated by the <resource-link/>.

- Place the layout tree after the component tree as shown.

```
<layout-tree>
  <layout-element uniqueName="com.ibm.isclite.samples.basicModule.layoutElement.A">
    <title>
      <base:nls-string lang="en">My basic module</base:nls-string>
    </title>
    <simple-container orientation="row"
      uniqueName="com.ibm.isclite.samples.basicModule.container.A">
      <window uniqueName="com.ibm.isclite.samples.basicModule.window.A">
        <title>
          <base:nls-string lang="en">My basic module</base:nls-string>
        </title>
        <component-definition-ref>
          com.ibm.isclite.samples.basicModule.portletEntity.A
        </component-definition-ref>
      </window>
    </simple-container>
  </layout-element>
</layout-tree>
```

- Each layout element in the layout tree defines the layout of a page in the console.
- Each page can have nested containers, and each simple container can have multiple windows. Each simple container has either a row or column orientation. The nesting of these containers can create a complex table structure for the page layout.
- The <component-definition-ref/> element specifies the unique name of the portlet entity that provides the content of this window.

- Create the navigation element after the layout tree as shown.

```
<navigation-element uniqueName="com.ibm.isclite.samples.navigationElement.basicModule"
  layoutElementRef="com.ibm.isclite.samples.basicModule.layoutElement.A">
  <title>
    <base:nls-string lang="en">Basic Module Sample</base:nls-string>
  </title>
  <preference name="ProductFilter">
    <base:value>
      <base:string>com.ibm.isclite.samples.PageLayout</base:string>
    </base:value>
  </preference>
  <parent-tree parentTreeRef="com.ibm.isclite.samples.navigationElement.Samples"/>
</navigation-element>
```

- Each navigation element defines a node in the console navigation. If you nest the navigation elements, it creates a hierarchical tree structure that is reflected in the console navigation. For the purpose of this exercise, only a single navigation element is provided. The **layoutElementRef** attribute indicates the unique name of the layout element that is rendered on the page that is displayed when this element is selected by the console user.
- The content of the **ProductFilter** preference indicates the **appid** of the lead application. This is used to render this navigation element when the lead product is selected by the console user for filtering the navigation.
- The content of the **parentTreeRef** element indicates the unique name of the parent navigation element for this element.

The following figure shows how this navigation element is rendered with the page layout sample. In this figure, the user has selected Integrated Solutions Console Sample from the **View** selection list to filter the navigation.

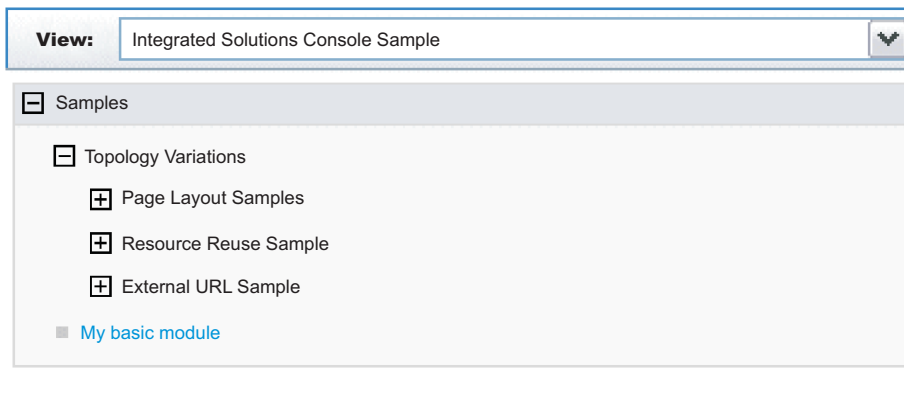


Figure 1. Sample console module navigation

What to do next

When you have finished, save and close the file and store in the /WEB-INF directory of your console module project.

Portal security descriptor: Before you begin

To follow these steps, open a file with the name `ibm-portal-security.xml` in a text or XML editor and save each change at the end of each step.

1. Copy the following elements into `ibm-portal-security.xml`. Some information is split on multiple lines for printing purposes.

```
<?xml version="1.0" encoding="UTF-8"?>

<ibm-portal-security
  xmlns="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-security.xsd"
  xmlns:base="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-base.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-topology.xsd
  ibm-portal-topology.xsd
    http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-base.xsd ibm-portal-base.xsd
    http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-security.xsd ibm-portal-security.xsd">

  </ibm-portal-security>
```

2. Add the `<application-role/>` element as the content of the `<ibm-portal-security/>` element.

```
<application-role uniqueName="monitor">

</application-role>
```

This sample identifies the administrative role, `monitor`, for access rights. See “Portal security schema” on page 111 for a complete description of this element and the `uniqueName` attribute.

3. Add the following `<portal-role/>` elements as the content of the application role.

```
<portal-role object-ref="com.ibm.isclite.samples.navigationElement.basicModule"/>
<portal-role object-ref="com.ibm.isclite.samples.basicModule.navigationElement.A"/>
<portal-role object-ref="com.ibm.isclite.samples.basicModule.portletEntity.A"
  role-type="Privileged User"/>
```

This sample assigns permission to a user with `monitor` access to view two navigation elements and to view, edit, and access help for one portlet. See “Portal security schema” on page 111 for a complete description of this element and its attributes.

What to do next

When you have finished, save and close the file and store in the /WEB-INF directory of your console module project.

Packaging a console module Before you begin

Before you start this task, be sure your Java sources compiles without errors and the XML descriptors have been validated against the corresponding schemas.

About this task

An Integrated Solutions Console module must be packaged as a WAR file for deployment. For best results, use a comprehensive development tool such as Rational Application Developer. Otherwise, you can follow these steps to package your console module resources into the WAR file.

1. Create a working directory to collect all of the files that need to be packaged in the WAR file.
2. Organize your portlets and other resource files in the prescribed directory structure for portlet WAR files using the following steps:
 - a. Place the following deployment descriptors in the directory /WEB-INF directory:
 - ibm-portal-topology.xml (topology descriptor)
 - ibm-portal-security.xml (security descriptor)
 - portlet.xml (portlet descriptor)
 - web.xml (Web application descriptor)
 - b. Place individual Java classes for your portlets in the directory /WEB-INF/classes. Use a directory structure that reflects the fully-qualified class name of your portlet, for example, /com/ibm/isclite/samples/basicModule.
 - c. Put JAR files in the directory /WEB-INF/lib.
 - d. If the console module supports multiple languages, place the properties file that contains the translated strings (the resource bundles) in the directory /WEB-INF/classes/nls.
 - e. Put any JavaServer Pages that must be served directly by the client in a directory that is not under the /WEB-INF directory (such /jsp). Resources in the /WEB-INF directory are protected and are not URL-addressable. JavaServer Pages that are accessed only by the portlets in the console module can be placed in the /WEB-INF path.
 - f. If the console module has help files, copy the Eclipse help plug-in files to the directory /help/plugin_name, where *plugin_name* is the name of the Eclipse help plug-in, such as com.ibm.myprod.help.doc.
3. Use the Java JAR utility to create the WAR file. For instructions, see the Java Tutorial at <http://java.sun.com/docs/books/tutorial/jar/basics/index.html> . You can also use an assembly tool like Rational Application Developer to create the WAR file for the console module.

Deploying a console module

After you create the WAR file for a console module, you can deploy the console module to a system where the runtime for Integrated Solutions Console is installed and then test that the console module works correctly. Only one instance of a console module can be deployed on an Integrated Solutions Console installation. This rule applies even if the versions of the console module are different.

About this task

If you need to update the version of a console module, you must perform these steps.

1. Remove the console module.
2. Update the module with the required changes.

3. Reissue the deploy command.

To deploy a console module on a system where the runtime for Integrated Solutions Console is installed, perform the following procedure:

1. Go to the machine where the Integrated Solutions Console runtime is installed. You can also use a machine that has remote access to the drive where the runtime is installed.
2. Copy the console module WAR file to the *app_server_root/systemApps/isclite.ear* directory. As an alternative, you can extract the WAR file into a subdirectory of this location using the WAR file name for the subdirectory. For example, you would extract *pagelayout.war* to the *app_server_root/systemApps/isclite.ear/pagelayout.war* directory.
3. Grant the QEJBSVR profile the authority needed to complete the task. From an i5/OS command prompt, issue this command:

```
CHGAUT OBJ('path_to_war') USER(QEJBSVR) DTAAUT(*RWX) OBJAUT(*ALL)
```

4. Start the wsadmin scripting client. If the server is not running, use the `-conntype NONE` option to run in local mode.
5. At the **wsadmin** command prompt, enter the command syntax. The following example presents the syntax:

```
$AdminApp update isclite modulefile {  
-operation add  
-contents path_to_war  
-contenturi war_filename  
-custom paavalidation=true  
-custom containWebServices=true|false  
-usedefaultbindings -contextroot /context  
-MapWebModToVH {{.* .* admin_host}}  
-MapModulesToServers {{.* *.war,.* WebSphere:cell=cellname,node=nodename,server=servername}}}
```

Note:

- The values of the `-contents` and `-contenturi` options are case sensitive and should match the war file name that you are ready to deploy.
- Specifying `-custom paavalidation=true` causes the deployment to validate the XML descriptors for the console module package before deployment. This is optional.
- Set the option `-custom containWebServices=true` only if the package contains a Web service. This setting invokes more deployment processing to locate and setup the Web service. The default for this option is true. If the package does not contain any Web services, be sure to set this option to false for more efficient deployment.
- Use the `-MapModulesToServers` option when deploying to an agent profile.

For example:

```
$AdminApp update isclite modulefile {  
-operation add  
-contents app_server_root/systemApps/isclite.ear/pagelayout.war  
-contenturi pagelayout.war  
-custom paavalidation=true  
-custom containWebServices=false  
-usedefaultbindings -contextroot /ibm/pagelayout  
-contextroot /pagelayout  
-MapWebModToVH {{.* .* admin_host}}  
-MapModulesToServers {{.* *.war,.* WebSphere:cell=cell_A,node=node_B,server=server_C}}}
```

The following messages are displayed if the deployment is successful.

```
Update of isclite has started.  
ADMA5009I: An application archive is extracted at profile_root/wstemp/wstemp/app_10793d84b8e/ext/pagelayout.war  
ADMA5064I: FileMergeTask completed successfully for isclite.  
ADMA5005I: The application isclite is configured in the WebSphere Application Server repository.  
ADMA5005I: The application isclite is configured in the WebSphere Application Server repository.
```

ADMA5110I: The application isclite is installed as a hidden application and will not be exposed via administrative interfaces such as GUI client, wsadmin or MBean Java API. In order to perform management operations on this application, the application name must be known.

CWLAA10001I: The Integrated Solutions Console module was deployed successfully.

ADMA5005I: The application isclite is configured in the WebSphere Application Server repository.

ADMA5011I: The cleanup of the temp directory for application isclite is complete.

Update of isclite has ended.

6. After the deployment has completed successfully, save your work by issuing the following command:

```
$AdminConfig save
```

Results

If the deployment is successful, the WAR file contents are extracted to the `/systemApps/isclite.war` directory and the module metadata is copied to the following directory:

```
profile_root/config/cells/cellname/applications/isclite.ear/deployments/isclite
```

To view your console module, you must log back in to the console. Make sure the navigation is updated correctly. If so, select the new navigation nodes verify that the new pages and console modules are rendered correctly.

Testing a console module

About this task

Use the following checklist to test your console module after it has been deployed.

1. Log in to Integrated Solutions Console as an administrator.
2. If the module includes an `<about-page/>` element in the topology descriptor, verify that your suite and its version are listed in the table on the Welcome page. Also confirm that the product is listed in the **View** selection list for the navigation.
3. Verify that the console navigation tree contains all of the organizational nodes and pages that were defined. Verify that each node is at the correct level in the tree.
4. For each page, verify that the portlets on the page work correctly. If a portlet on the page launches a new page in work area, verify that the launching works correctly.
5. To test the access permissions specified in the console module descriptor, logout and then log in as a different user to verify that the correct pages are accessible.
6. If the console module included a help plug-in, make sure you can launch the help.
7. If the console module contains errors, refer to the `SystemOut.log` and `SystemErr.log` for any errors or messages. These logs are located in the following directory:

```
profile_root/logs/server1
```

Removing a console module

About this task

Follow these steps to delete a console module.

1. Ensure that the server is started. You do not need to log in to the console.
2. Start the wsadmin scripting client.
3. Enter the following command from the wsadmin command prompt.

```
$AdminApp update isclite modulefile {-operation delete -contenturi module_file_name.war}
```
4. Save your configuration changes.

```
$AdminConfig save
```
5. Exit the wsadmin command environment.

quit

Adding advanced API features

The class files for console modules are developed using the Java Portlet Specification. Integrated Solutions Console includes additional APIs for launching pages, passing properties to other portlets, and launching Eclipse-based help. This section provides information about the APIs available to console modules.

Developing portlets

Console modules are portlets that run in the portlet container of WebSphere Application Server.

Integrated Solutions Console supports console modules written to the Java Portlet Specification. A packaged console module WAR file contains one or more portlets or reuses portlets from other modules. According to the JSR 168 specification, a *portlet* is a Java technology based Web component, managed by a portlet container, that processes requests and generates content that contributes to the overall portal page. Portlets share many of the same characteristics as servlets. However, portlets also exhibit differences. For example, portlets cannot set HTTP headers and can contribute only markup fragments to the portal page. The portlet specification leverages the functionality of the servlet specification as much as possible, and many parts of the Java Portlet Specification have been modeled after the servlet specification.

The application server includes a JSR 168-compliant portlet container to support deploying and running console modules in Integrated Solutions Console. The JSR 168 portlet container relies on the Java Platform, Enterprise Edition (Java EE) architecture implemented by the application server. Consequently, an Integrated Solutions Console module and its portlets are packaged and deployed similar to Java EE Web applications. In addition to the Web application descriptor and the portlet descriptor, the WAR file for an Integrated Solutions Console module includes additional descriptors needed to deploy the application to a console installation.

The following Web resources can be useful for learning more about portlet development.

Resource	Web site URL
Portlet Specification 1.0	http://jcp.org/jsr/detail/168.jsp
Portlet Specification 2.0	http://jcp.org/en/jsr/detail?id=286
WebSphere Portal Zone	http://www.ibm.com/developerworks/websphere/zones/portal/
<i>Best practices: Developing portlets using JSR 168 and WebSphere Portal</i>	http://www.ibm.com/developerworks/websphere/library/techarticles/0403_hepper/0403_hepper.html
WebSphere Portal external newsgroup	news://news.software.ibm.com/ibm.software.websphere.portal-server
<i>Develop a Faces JSR 168 portlet using IBM Rational Application Developer 6.0 for autonomic computing</i>	http://www.ibm.com/developerworks/edu/ac-dw-ac-jsfisc-i.html
<i>Introduction to JSR 168 - The Portlet Specification</i>	http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/index.html

Launching pages

Typically, users of the console launch pages using the links in the console navigation. However, portlets in a console module also launch pages directly using the Dynamic UI Manager API.

The DynamicUI Manager APIs are provided by 3 services: PropertyFactory, DynamicUIManagerFactoryService, and URLGeneratorFactoryService. To use these services the portlet must first perform a JNDI lookup to get the PortletServiceHome object which you then can use to get the

actual service interface object. This is demonstrated in the GetApps portlet of the PortletContext sample.. The following code is placed in a try block of the portlet's init() method.

Some information is split on multiple lines for printing purposes.

```
Context ctx = new InitialContext(com.ibm.portal.jndi.Constants.ENV);
// get propertyfactory service

propertyFactoryServiceHome = (PortletServiceHome)ctx.lookup("iscportletservice/com.ibm.portal.
propertybroker.service.PropertyBrokerService");
if (propertyFactoryServiceHome == null)
    logger.log(Level.FINE, "no PropertyBrokerService propertyFactoryServiceHome");
else
    propertyFactoryService =
        (PropertyFactory)propertyFactoryServiceHome.getPortletService(com.ibm.portal.
propertybroker.service.PropertyBrokerService.class);
logger.log(Level.FINE, "propertyBrokerService="+propertyFactoryService);
// get dynamicUIManagerFactory service
dynamicUIManagerFactoryServiceHome =
    (PortletServiceHome)ctx.lookup("iscportletservice/com.ibm.portal.portlet.service.DynamicUIManagerFactoryService");
if (dynamicUIManagerFactoryServiceHome == null)
    logger.log(Level.FINE, "no dynamicUIManagerFactoryService propertyFactoryServiceHome");
else
    dynamicUIManagerFactoryService =
        (DynamicUIManagerFactoryService)dynamicUIManagerFactoryServiceHome.getPortletService
(com.ibm.portal.portlet.service.DynamicUIManagerFactoryService.class);
logger.log(Level.FINE, "dynamicUIManagerFactoryService="+dynamicUIManagerFactoryService);
// get urlGeneratorFactory service
urlGeneratorFactoryServiceHome =
    (PortletServiceHome)ctx.lookup("iscportletservice/com.ibm.portal.portlet.service.URLGeneratorFactoryService");
if (urlGeneratorFactoryServiceHome == null)
    logger.log(Level.FINE, "no urlGeneratorFactoryService propertyFactoryServiceHome");
else
    urlGeneratorFactoryService =
        (URLGeneratorFactoryService) urlGeneratorFactoryServiceHome.getPortletService
(com.ibm.portal.portlet.service.URLGeneratorFactoryService.class);
logger.log(Level.FINE, "urlGeneratorFactoryService="+urlGeneratorFactoryService);
```

A portlet can launch a new page using the dynamicUICtrl interface, passing the object ID of the page definition for the new page. The object ID is determined by performing a JNDI lookup on the unique name of the navigation element that launches the page. If you are planning to launch a navigation element, the navigation node must point to a layout element. In other words the navigation element must contain the **layoutElementRef** attribute. The addPage() method returns the object ID of the page instance, which should be saved by the portlet to create a URL and launch the page. The following excerpt from the PortletContext sample demonstrates how this works. The portlet GetApps.java includes a launchPage() method with this code.

```
Context ctx;
ObjectID pageDefinitionID = null;
String pagename="com.ibm.isclite.portletcontext.navigationElement.B";
logger.log(Level.FINE, "launch page=" + pagename);

PortletSession ps = request.getPortletSession(false);
try {
    ctx = new InitialContext();
    pageDefinitionID =(ObjectID)ctx.lookup("portal:uniquename/"+pagename);
} catch (NamingException ne) {
    return;
}
try {
    PropertyController cproperty = propertyFactoryService.createProperty(myconfig);
    PropertyController cproperty2 = propertyFactoryService.createProperty(myconfig);
```

```

cproperty.setType("String");
cproperty.setName("servername");
cproperty2.setType("String");
cproperty2.setName("appDeployed");
PropertyValue[] propertyValues = new PropertyValue[2];
propertyValues[0] =
    propertyFactoryService.createPropertyValue(request, cproperty, serverName);
propertyValues[1] =
    propertyFactoryService.createPropertyValue(request, cproperty2, appDeployed);

DynamicUICtrl dmanagerCtrl =
    dynamicUIManagerFactoryService.getDynamicUICtrl(request, response, "isc.tasklaunch");

ObjectID newPageID =
    dmanagerCtrl.addPage(pageDefinitionID, null, propertyValues);

RedirectURLGenerator urlGenerator =
    urlGeneratorFactoryService.getURLGenerator(request, response);

EngineURL redirectURL = urlGenerator.createPageURL(newPageID);

response.sendRedirect(redirectURL.toString());
} catch ...

```

- The pagename String is created with the unique name of the navigation element (from the topology descriptor) that launches the page.
- The object ID is found by performing a JNDI lookup on this Context instance using this String:
"portal:uniquename/"+pagename

The String portal:uniquename/ is required to obtain the object ID for the navigation element.

- Properties are created to be passed to the portlets on the page. The property names are servername and appDeployed.
- In addition to the request and response, you must include the String isc.tasklaunch as the configuration name on the getDynamicUICtrl() method.
- Use the addPage() method of the new DynamicUICtrl instance to create and instance of the page. Any properties that need to be passed to portlets on the new page must be provided in this method. The addPage() method does not launch the page, but it returns the object ID of the page instance. The next two methods, getURLGenerator() and createPageURL(), are used to construct the URL that opens the page. The URL contains property names and values if they have been provided.
- The URL to open the page is provided on the sendRedirect() method of the response.

Passing properties to other portlets

This topic describes how portlets can exchange properties. You should be familiar with the concepts for launching console pages before reading this topic.

Information can be provided to portlets on the same page using the addSharedPortlet() method. Property values are of the type com.ibm.portal.propertybroker.property.PropertyValue. The example in “Launching pages” on page 92 shows these properties passed to the page using an instance of the PropertyValue interface (propertyValues) and shows how that object was created using the PropertyFactory interface.

The SetServer.java source from the PortletContext sample shows how to pass properties to a portlet on the same page. It is assumed that the portlet has already performed a JNDI lookup to determine that the PropertyFactory, DynamicUIManagerFactoryService, and URLGeneratorFactoryService services are available. This is described in “Launching pages” on page 92

- In this example, the portlet that is the source of the properties needs the object ID of the target portlet. The object ID is determined by performing a JNDI lookup using the values from the portlet’s <resource-link/> element in the topology descriptor.
- The property, cproperty, is created using the PropertyController interface and the createProperty() method of the PropertyFactory interface. The data type for the property must be a String.

- Prior to sending the property, the source portlet creates an instance of the DynamicUICtrl interface, passing the string "isc.tasklaunch" as the configuration name.
- The addSharedPortlet() method sends the properties to the target portlet on the request. After the action phase, the target portlet updates the output for the response.

```
Context ctx = null;
ObjectID portletDefinitionID1 = null;
ObjectID portletDefinitionID2 = null;

String portletname1="com.ibm.isclite.samples.PortletContext/PortletGetPerformance";
String portletname2="com.ibm.isclite.samples.PortletContext/PortletGetApps";

PortletSession ps = request.getPortletSession(false);

try
{
    ctx = new InitialContext(com.ibm.portal.jndi.Constants.ENV);
    portletDefinitionID1 =
        (ObjectID)ctx.lookup("portal:config/portletdefinition/"+portletname1);
    portletDefinitionID2 =
        (ObjectID)ctx.lookup("portal:config/portletdefinition/"+portletname2);
} catch (NamingException ne)
{
    logger.log(Level.FINE, "portletdefinitionID not found - Naming exception:"+ne.getMessage());
    return;
}
logger.log(Level.FINE, "portletdefinitionID="+portletDefinitionID1.toString());

try
{
    PropertyController cproperty =
        propertyFactoryService.createProperty(myconfig);
    cproperty.setType("String");

    PropertyValue[] propertyValues = new PropertyValue[1];
    propertyValues[0] =
        propertyFactoryService.createPropertyValue(request, cproperty, serverName);

    DynamicUICtrl dmanagerCtrl =
        dynamicUIManagerFactoryService.getDynamicUICtrl(request, response, "isc.tasklaunch");

    ObjectID newPortletID1 =
        dmanagerCtrl.addSharedPortlet(portletDefinitionID1, null, propertyValues);
    ObjectID newPortletID2 =
        dmanagerCtrl.addSharedPortlet(portletDefinitionID2, null, propertyValues);

    logger.log(Level.FINE, "portlet ID created:"+newPortletID1.getOID());
} catch ...
```

In some cases, the target portlet might be on a separate page. In this case, the properties are passed using the addPage() method. The target portlet receives the properties only when the page is launched that contains the portlet. If a property value is set multiple times before the page is launched, the value that was set last for the property is passed to the portlets on the page. You can use the URLGeneratorFactoryService interface to redirect the console to the page for the target portlet. This is demonstrated in the GetApps portlet of the PortletContext sample.

```
ObjectID newPageID = dmanagerCtrl.addPage(pageDefinitionID, null, propertyValues);
RedirectURLGenerator urlGenerator = urlGeneratorFactoryService.getURLGenerator(request, response);
EngineURL redirectURL = urlGenerator.createPageURL(newPageID);
response.sendRedirect(redirectURL.toString());
```

To use the URLGeneratorFactoryService, the portlet must first perform a JNDI lookup for this portlet service. Refer to the GetApps.java source to see how to obtain a reference to the portlet service.

Receiving properties

To receive properties, the target portlet must provide the `com.ibm.portal.pagecontext.enable` preference parameter in the `portlet.xml` with a value of `true`. If the portlet should receive any subsequent updates, the `com.ibm.portal.context.enable` read-only preference should also be set to `true`. Only String property types are supported and the context is passed as parameters of the action request. The following example shows how the `GetApps` portlet receives properties passed by the `SetServer` portlet.

```
public void processAction(ActionRequest request, ActionResponse response)
    throws PortletException, IOException {

    PortletSession ps = request.getPortletSession(true);
    appDeployed = request.getParameter("appDeployed");
    serverName=request.getParameter("servername");

    ps.setAttribute("servername",serverName);
    ps.setAttribute("appDeployed",appDeployed);
    launchPage(request, response);

}
```

A target portlet can check `com.ibm.portal.action` parameter of the request during action processing to determine if any properties have been passed. If properties are being passed to the portlet, the value of this parameter is `com.ibm.portal.pagecontext.receive`. For example:

```
String action = req.getParameter(com.ibm.portal.action.name);
if (action!=null && action.equalsIgnoreCase("com.ibm.portal.pagecontext.receive")) {


    // code to get the properties as a parameter on the request

}
```

Launching Eclipse-based help

If your console module contains a portlet that requires a help page, follow the instructions in this topic to implement help for the portlet.

About this task

To implement portlet help, you use the portlet's `doHelp()` method, the `EclipseHelp` class, and the help setting in the `portlet.xml` file. Inside the `doHelp()` method, link to the target help topic in your Eclipse help plug-in by using the `portletHelp()` method of the `EclipseHelp` class. By including a `doHelp()` method in your portlet and adding the help setting to the `portlet.xml` file, you enable the portlet help mode. When portlet help mode is enabled, the  icon is displayed on the portlet title bar. When a user clicks the icon, a separate browser window opens and displays only that help topic.

The Modes sample (`modes.war`) demonstrates how a console module can provide user assistance or help. This sample is described in "Example: Console module samples" on page 82.

1. Add the `doHelp()` method to your portlet.
 - a. In the portlet Java file, include the following import statement:

```
import com.ibm.portal.help.EclipseHelp;
```

- b. Implement the `doHelp()` method in the portlet.

```
public void doHelp(RenderRequest request, RenderResponse response)
    throws PortletException, IOException {

}
```

- c. Create a String in the `doHelp()` method to contain the help topic. The syntax for specifying a help topic is `pluginID/file_path/file_name`, where `pluginID` is the Eclipse help plug-in ID as specified in its `plugin.xml` file, `file_path` is the sub-directory path (if any), and `file_name` is the filename and extension of the help content file.

```

public void doHelp(RenderRequest request, RenderResponse response)
    throws PortletException, IOException {

    String topic = "com.ibm.isc.help.modes/help_mode.html";
}

```

- d. Call the method `EclipseHelp.portletHelp()` passing in the request, response, and topic as shown in the code snippet below.

```

public void doHelp(RenderRequest request, RenderResponse response)
    throws PortletException, IOException{

    String topic = "com.ibm.isc.help.modes/help_mode.html";
    EclipseHelp.portletHelp(request, response, topic);

}

```

2. Set help mode in the `portlet.xml` file.
 - a. Use a text editor to open the portlet deployment descriptor (the `portlet.xml` file). See topics under Developing portlets for more information about creating a `portlet.xml` file.
 - b. Inside the `<portlet-app>` element, locate the `<portlet>` element for the portlet that implements the `doHelp()` method.
3. Inside the `<supports>` sub-element, add a `<portlet-mode>` element like the one shown in the following example.

```

<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
    version="1.0" id="com.ibm.isclite.samples.Modes"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
        http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
    <portlet id="Modes_Portlet" >
        <portlet-name>ModesPortlet</portlet-name>
        <display-name>Modes portlet1 (JSR 168)</display-name>
        <display-name xml:lang="en">Modes portlet1 (JSR 168)</display-name>
        <portlet-class>com.ibm.isclite.samples.Modes</portlet-class>
        <expiration-cache>0</expiration-cache>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
            <portlet-mode>EDIT</portlet-mode>
            <portlet-mode>HELP</portlet-mode>
        </supports>
        <supported-locale>en</supported-locale>
        <resource-bundle>nls.HTML</resource-bundle>
    </portlet>

</portlet-app>

```

4. When packaging the console module, place the Eclipse help plugin in the `/help` directory of the WAR file. See “Packaging a console module” on page 89 for more information.

Support for bidirectional characters

Integrated Solutions Console supports bidirectional text. If your console module provides support for Hebrew, it must provide two identical resource bundles for the translated Strings. The file name for one resource bundle must be appended with the `_iw` suffix and the other resource bundle must be appended with the `_he` suffix. For example:

```

portlet_iw.properties
portlet_he.properties

```

Console module schemas

Descriptors for a console module

Console modules are packaged as *portal application archives* (PAA). A portal application archive is similar to a standard JSR 168 portlet WAR file - it includes one or more portlets plus the servlet and portlet descriptors (web.xml and portlet.xml) However, the portal application archive includes two additional descriptors:

ibm-portal-topology.xml

Defines the console module and its components, page layout, and location in the navigation. Based on the schema definitions in `ibm-portal-topology.xsd`.

ibm-portal-security.xml

Describes portal application security, such as application roles and access control. Based on the schema definitions in `ibm-portal-security.xsd`.

When a console module is deployed, Integrated Solutions Console reads these descriptors and uses that information to add the console module to the installation. The XML descriptor files must be well-formed and validate with the associated schemas. If the file is not valid, deployment ends and an error message is generated.

A copy of the schema files (with descriptive comments embedded) is in the `app_server_root/properties/schemas` directory. Not all elements and attributes in the schemas are supported, however. Only elements and attributes that are documented in the following sections are supported:

- “Base schema”
- “Portal topology schema” on page 99
- “Application definition elements” on page 101
- “Portal security schema” on page 111

Base schema

The topology and security schemas reference the base schema for portal application archives, `ibm-portal-base.xsd`. The following complex type element definitions from the base schema are used most often in the topology and security descriptors. These elements can be used within any other element in the descriptor that requires a localized string to be displayed, for example, in the title or description elements.

NlsString

specifies localized strings to be displayed. This tag is intended to be used during development before the application has been updated to support multiple languages through resource bundles. The `xml:lang` attribute enables a programmer to get started with the code without having to define a resource bundle right at the start. The value of the `xml:lang` attribute should adhere to RFC 1766 (<http://www.ietf.org/rfc/rfc1766.txt>). The default value of the `xml:lang` attribute is English(en).

```
<base:nls-string lang="locale">string</base:nls-string>
```

NlsRef

specifies localized strings that are maintained in a Java resource bundle. You must include the location of the resource bundle and the name of the key for lookup.

```
<base:nls-ref key="key_name" locationName="path_to_resource_bundle"/>
```

where `path_to_resource_bundle` specifies a path relative to the `/WEB-INF` directory of the module's WAR file. For example, if the resource bundle is located in `/WEB-INF/classes/nls/myresources.properties`, specify `/classes/nls/myresources`.

If the resource bundle cannot be located, the key name is returned. That is, if the property string that could not be found is intended to be displayed in the console, the key name would be displayed instead.

Preference

Used to describe name/value pairs used by the application. The preference name is specified as an attribute and the preference value is specified as a subelement.

```
<preference name="preference_name">
  <base:value>
    <base:string>preference_value</base:string>
  </base:value>
</preference>
```

The preference value can be specified as a string, NlsString, or NlsRef.

Displaying version information in the Welcome portlet

To display the minor version of a product in the Welcome portlet without redeploying, add a Preference attribute to the `ibm-portal-topology.xml` file of your product. Then provide your own JSP to output the version string. Your JSP will be able to access any class in any of the WAS-embedded JAR files and return the version string.

This method will supersede the `versionString` specified under ApplicationDefinition element in the `ibm-portal-topology.xml` file. That method of specifying the version number is also supported.

When using the `ibm-portal-topology.xml` file to display the minor version of a product, add the preference attribute with `name="versionInfoJSP"`. When you subsequently deploy your application, the `component.xml` file will be updated to include this information as described below. You would also need to create a JSP, using any name, to contain the version number.

For example, if your JSP is under `abc.war/jsp/xyz.jsp`, the `ibm-portal-topology.xml` file would read as follows:

```
<preference name="versionInfoJSP">
  <base:value>
    <base:string>jsp/xyz.jsp</base:string>
  </base:value>
</preference>
```

The `component.xml` file would read as follows:

```
<registry:preference name="versionInfoJSP">
  <base:value>
    <base:string>jsp/xyz.jsp</base:string>
  </base:value>
```

The contents of `xyz.jsp` might read as follows:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page session="false" buffer="none"%>
1.0.0
```

Portal topology schema

The portal topology descriptor distinguishes the console module from other portlet applications by providing additional information about the portlets in the application. Whereas the `portlet.xml` provides a unique identifier for each portlet, its fully-qualified class name, and supported modes, the portal topology descriptor describes a console module in a generic sense, including a list of component portlets, the page layout, and navigation information. Portlets in the application can be packaged in the WAR file or already deployed from another archive. The only mapping between the descriptors is a reference from the portal application's `<resource-link/>` element, using the portlet application ID and portlet name from the portlet descriptor.

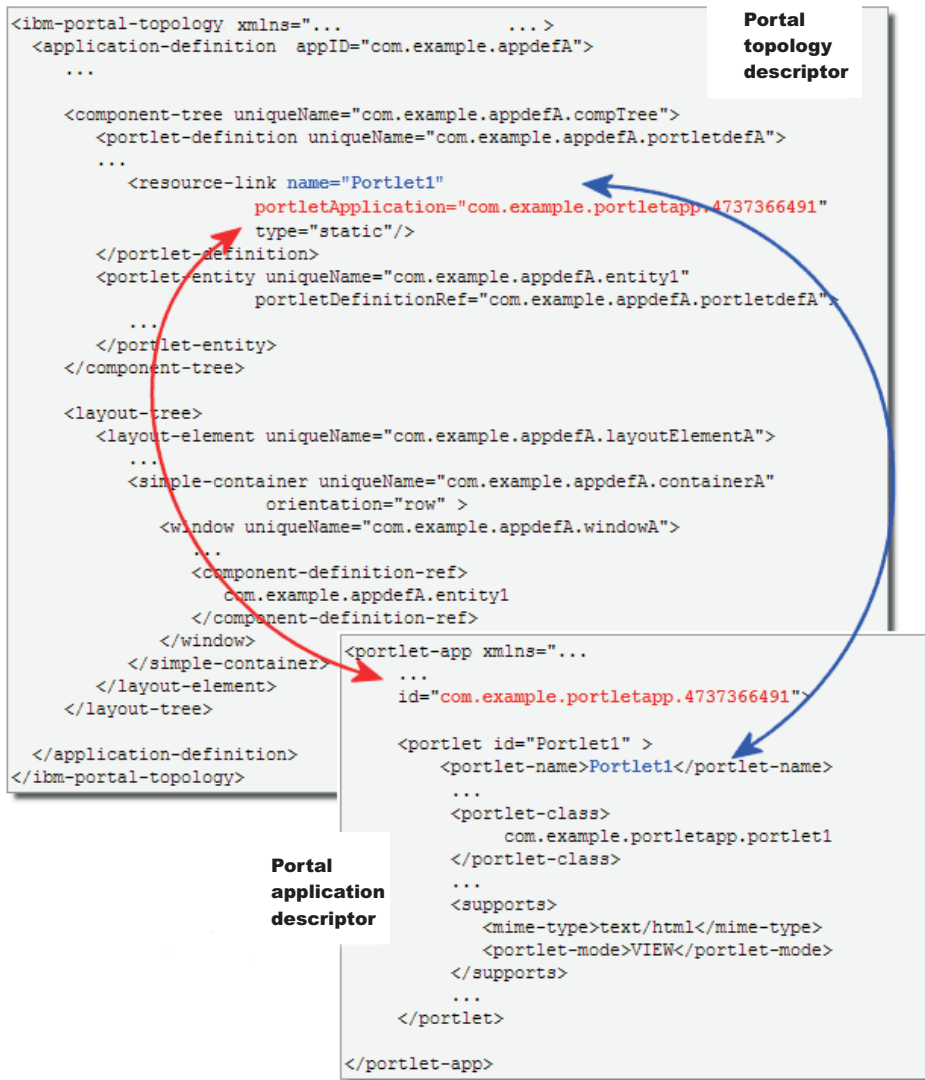


Figure 2. References from the portal topology descriptor to the portlet descriptor

Description of the top level topology elements

Only the following elements, along with those described in “Application definition elements” on page 101, are supported for this release.

<ibm-portal-topology/>

Required. the root element of the portal topology descriptor. The content of this tag is one or more `<application-definition/>` tags and, optionally, either an `<about-page/>` or a `<PAA-ref/>` element.

<application-definition/>

At least one is required. This element describes an application with its component portlets, page layout, and portal navigation structure. See Application definition elements.

After the application definition, only one of the following elements can be defined.

<about-page/>

Optional. The about page is provided by the lead application for a product and provides a description of the product and other resources for support. This content is specified using the

unique name of the layout element in the application definition that contains the about page content. At runtime, the about page is accessed from the main welcome page of the console. Providing the <about-page/> element also causes the console module to appear in the View selection list of the navigation.

In the following example from the page layout sample, about page refers to a layout element with the unique name `com.ibm.isclite.pagelayout.layoutElement.F`.

```
<about-page>com.ibm.isclite.pagelayout.layoutElement.F</about-page>
```

The layout element provided in this sample includes a portlet with the about page content.

<PAA-ref/>

Optional. This element references the **appid** of the lead application for the product. The **appid** name should be unique in the install. This element is intended for subcomponents of an application that spans multiple archives.

Application definition elements:

The <application-definition/> element describes the portal application, its components, page layout, and navigation. It is contained by the <ibm-portal-topology/> element in the portal topology descriptor.

Basic structure of the application definition

The following figure illustrates the elements of the application definition that are supported for this release.

```

<application-definition appID="" version="">

  <title/>

  <requires uniqueName="" version=""/>

  <component-tree uniqueName="">
    <portlet-definition uniqueName="">
      <title/>
      <resource-link name="" type="" portletApplication=""/>
    </portlet-definition>
    <portlet-entity uniqueName="" portletDefinitionRef="">
      <title/>
    </portlet-entity>
    <url-link uniqueName="" url="" absolute="" />
  </component-tree>

  <layout-tree>
    <layout-element uniqueName="">
      <title/>
      <simple-container uniqueName="" orientation="">
        <window uniqueName="">
          <title/>
          <component-definition-ref/>
        </window>
      </simple-container>
    </layout-element>
  </layout-tree>

  <navigation-element uniqueName="" layoutElementRef="">
    <title/>
    <preference>
      <value/>
    </preference>
    <url-mapping/>
    <parent-tree parentTreeRef="" ordinal=""/>
    <navigation-content uniqueName="" componentDefinitionRef="" windowRef=""/>
  </navigation-element>

</application-definition>

```

Figure 3. Elements in the portal application definition

Other elements of the application definition that can be found in the `ibm-portal-topology.xsd` schema are not supported for this release. At a minimum, the application definition requires a title, a component tree, and a layout tree. The following sections describe the `<application-definition/>` element, its attributes, and its subelements.

Application definition

The `<application-definition/>` element defines one or more applications that are part of the package. Exactly one is required.

The following attributes are used with this element.

appID Required. Specifies a unique identifier for this portal application. The appID has no relationship with IDs in the `portlet.xml`. To guarantee uniqueness, see “Console module elements - guidelines for unique identifiers” on page 111.

version

Required. Specifies the numerical version of the application. The format of the version is defined in the `ibm-portal-base.xsd` schema under the simple type for `VersionString`.

Title

Exactly one is required. The <title/> element provides a title for the application. If this is the lead application for a product, this title is displayed on the Welcome page when the user logs into the console. A lead application is the one that includes the <about-page/> element in the topology definition. Subcomponents of the product use the <PAA-ref/> element to specify the lead application to which they belong.

The content of the <title/> element can be specified as a simple string element from the base schema. To provide a localized string, use the <base:nls-string/> or <base:nls-ref/> elements for the title content. For example:

```
<title>
  <base:nls-string lang="en">Integrated Solutions Console Sample</base:nls-string>
</title>
```

Requires

Optional, multiple allowed. The <requires/> element indicates the unique ID of a portal application that is a prerequisite for this application. Any prerequisites for the application must be already deployed. The application can have multiple prerequisites.

The following attributes are used with this element.

uniqueName

Required. Specifies the unique identifier of another application that must already be deployed on the server. This value must match the appID attribute from the <application-definition/> element of the required application's topology descriptor. Otherwise, deployment fails.

version

Required. Specifies the numerical version from the required application's <application-definition/> element. If the version is not identical to the required application's version string, deployment fails. Required.

The following example shows how the tree merge sample indicates the page layout sample as a prerequisite application.

```
<requires uniqueName="com.ibm.isclite.samples.PageLayout" version="6.1"/>
```

The following example shows the attributes for the <application-definition/> element of the page layout sample. These attributes are used as the values for attributes on the <requires/> element for the tree merge sample.

```
<application-definition appID="com.ibm.isclite.samples.PageLayout" version="6.1">
...
```

Component tree

Exactly one is required. The <component-tree/> element describes all portlets in the console module. The tree structure of the component definitions can be used for organizing the components in the portal application. There are no semantics associated with this organization and the portal may not respect this structure for organizing the actual portal resources corresponding to it.

The following attribute is used with this element.

uniqueName

Required. Specifies a unique identifier for the component tree. To guarantee uniqueness, see "Console module elements - guidelines for unique identifiers" on page 111.

The following elements make up the content of the <component-tree/> element.

<portlet-definition/>

Optional, multiple allowed. This element describes one portlet. The portlet can be included in the same archive or already deployed on the server.. A portlet definition refers to a portlet defined in a portlet.xml deployment descriptor in a portlet WAR file. A portlet can be associated with a window in a layout. In most cases, the application artifact is actually defined by its own deployment descriptor within the archive and the <resource-link/> element is used for referencing it so that it can be integrated into the portal application page layout and navigation.

The following attribute is used with this element.

uniqueName

Required. Specifies a unique identifier for the portlet definition. To guarantee uniqueness, see “Console module elements - guidelines for unique identifiers” on page 111.

The following element makes up the content of the <portlet-definition/>.

<title/>

Required, exactly one allowed. This element provides a title for the portlet definition. This title is not displayed in the console for this release. To provide a localized string, use the <base:nls-string/> or <base:nls-ref/> elements for the title content.

<resource-link/>

Required, exactly one allowed. This element provides a link to a portlet definition that is included in the PAA package or is already deployed separately in the portal.

The following attributes are used with this element.

name Specifies the name of the portlet as defined by the <portlet-name/> element in the portlet.xml. If the name of the portlet is not unique within the PAA, an additional portlet-application attribute must be specified.

type The value `static` specifies that the resource must be available at the time the PAA package is deployed. This is the default and the only value that is supported for this release.

portletApplication

Specifies the ID attribute of the <portlet-app/> element in the portlet.xml. Since different PAA packages could have portlets with the same name, it is necessary to name space them.

The following example shows the <resource-link/> element for the page layout sample.

```
<resource-link name="PageLayoutPortlet1"
  portletApplication="com.ibm.isclite.samples.PageLayout" type="static"/>
```

The following example is taken from the portlet.xml for the page layout sample to show the values needed for the previous <resource-link/> example.

```
...
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
  version="1.0" id="com.ibm.isclite.samples.PageLayout"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
    http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
  <portlet id="PageLayout_Portlet1" >
    <portlet-name>PageLayoutPortlet1</portlet-name>
  ...
```

The next example is also from the page layout sample. In this example, the <resource-link/> uses values from a portlet that is already deployed to the portal. These values come from the portlet.xml of the portlet (TempPortlet) from the tree merge sample.

```
<portlet-definition uniqueName="com.ibm.isclite.TreeMerge.appElementDefinition.A">
  <title>
    <base:nls-string lang="en">TreeMergePortlet</base:nls-string>
  </title>
  <resource-link name="TempPortlet"
```

```

        portletApplication="com.ibm.isclite.samples.TreeMerge" type="static"/>
    </portlet-definition>
    <portlet-entity uniqueName="com.ibm.isclite.TreeMerge.appElement.B"
        portletDefinitionRef="com.ibm.isclite.TreeMerge.appElementDefinition.A">
        <title>
            <base:nls-string lang="en">TempPortlet</base:nls-string>
        </title>
    </portlet-entity>

```

<portlet-entity/>

Optional, multiple allowed. This element specifies an instance of a portlet definition with additional configuration values. The portlet entity's uniqueName is used by the <window/> element to place the portlet on the page. The portlet definition that is specified can reside in the current archive or already be deployed on the server.

The following attributes are used with this element.

uniqueName

Required. Specifies a unique identifier for the portlet entity. To guarantee uniqueness, see "Console module elements - guidelines for unique identifiers" on page 111.

portletDefinitionRef

Required. References the unique name of the portlet definition within the same topology descriptor file.

The following element makes up the content of the portlet definition.

<title/>

Optional, no more than one allowed. This element provides a title for the portlet entity. To provide a localized string, use the <base:nls-string/> or <base:nls-ref/> elements for the title content. This title is rendered in the portlet window.

<url-link/>

Optional and multiple elements are allowed. This element is used to include links to external URLs that are added to the console navigation . The following attributes are used with this element.

uniqueName

Required. Specifies a unique identifier for the url link. To guarantee uniqueness, see "Console module elements - guidelines for unique identifiers" on page 111.

absolute

Optional. Indicates whether the URL is an absolute value (true) or relative value (false). Only absolute values are supported for this release and the default value is true.

url Required. Specifies an external address or URL that will be added to the console navigation.

In the following example, the URL for a company support page is provided.

```

<url-link uniqueName="my.example.com.support.page" url="http://my.example.com/support" absolute="true"/>

```

The following example shows part of the <component-tree/> element for the page layout sample with the first portlet definition and corresponding portlet entity.

```

<component-tree uniqueName="com.ibm.isclite.pagelayout.appTree">
    <!-- PageLayoutPortlet 1-->
    <portlet-definition uniqueName="com.ibm.isclite.pagelayout.appElementDefinition.A">
        <title>
            <base:nls-string lang="en">RowPortlet</base:nls-string>
        </title>
        <resource-link name="PageLayoutPortlet1"
            portletApplication="com.ibm.isclite.samples.PageLayout" type="static"/>
    </portlet-definition>
    <portlet-entity uniqueName="com.ibm.isclite.pagelayout.appElement.A"
        portletDefinitionRef="com.ibm.isclite.pagelayout.appElementDefinition.A">

```

```
<title>
  <base:nls-string lang="en">RowPortlet</base:nls-string>
</title>
</portlet-entity>
```

...

Layout tree

Exactly one is required. Each layout element describes the layout for a page that is opened by a navigation element. There is a one to one correspondence between a page and a layout element. The page layout is determined by row and column containers. Each row or column can contain any number of windows, which in turn contain one portlet each. The layout elements, containers, and windows can be fully defined, including their content, or they can be empty templates to be filled when content is added subsequently.

The following element makes up the content of the layout tree.

<layout-element/>

Optional and multiple allowed. The layout element describes the layout of a portal page. Provide one layout element for each page that you want to define in the application.

The following attribute is used with this element.

uniqueName

Required. Specifies a unique identifier for the layout element. To guarantee uniqueness, see “Console module elements - guidelines for unique identifiers” on page 111.

The following elements make up the content of the layout element.

<title/>

Exactly one is required. This element provides a title for the layout element. To provide a localized string, use the `<base:nls-string/>` or `<base:nls-ref/>` elements for the title content. This title is displayed as the title for the corresponding page.

<simple-container/>

At least one is required. This element defines a row or column in the page layout. This element can be nested.

The following attributes are used with this element.

uniqueName

Required. Specifies a unique identifier for the layout element. To guarantee uniqueness, see “Console module elements - guidelines for unique identifiers” on page 111.

orientation

Required. Specifies how the contents of the container are arranged. The value `row` arranges content from left to right. The value `column` arranges content from top to bottom.

The following element makes up the content of this element.

<window/>

Optional and multiple are allowed. A window describes a place in the page layout where portlet content can be inserted. The window can either contain an application artifact or a nested layout.

The following attribute is used for the window.

uniqueName

Required. Specifies a unique identifier for the window. To guarantee uniqueness, see “Console module elements - guidelines for unique identifiers” on page 111.

The following elements make up the content of this element.

<title/>

Optional, no more than one allowed. To provide a localized string, use the <base:nls-string/> or <base:nls-ref/> elements for the title content. This string is rendered in the portlet title bar if the title for the <portlet-entity/> is not provided. If neither the portlet entity or the window provide a title, the title is obtained from the portlet.xml (<portlet-name/> element) of the portlet.

<component-definition-ref/>

Optional, but no more than one allowed. This element specifies a reference to the unique name of the component definition for the portlet to be included in this window.

The page layout sample contains six layout elements in its layout tree. This provides six pages showing different ways to arrange the content on the page. The following example shows the first layout element with its simple containers and windows. The inline comments are also shown.

<layout-tree>

```

<!--PageLayout A contains Two Seperate Containers:
  The first Container has row orientation with one window inside the container
  The second Container has column orientation with one window inside the container

  This layout uses is-spot to specify the application for the window.
  Therefore you must specify the navigation-content in the navigation element
  below that uses this layout element.
-->
<layout-element uniqueName="com.ibm.isclite.pagelayout.layoutElement.A">
  <title>
    <base:nls-string lang="en">Two independent Containers</base:nls-string>
  </title>
  <simple-container orientation="row"
    uniqueName="com.ibm.isclite.pagelayout.container.A1A">
    <simple-container orientation="row"
      uniqueName="com.ibm.isclite.pagelayout.container.A1">
      <window uniqueName="com.ibm.isclite.pagelayout.window.A1">
        <title>
          <base:nls-string lang="en">Row Portlet</base:nls-string>
        </title>
        <component-definition-ref>
          com.ibm.isclite.pagelayout.appElement.A
        </component-definition-ref>
      </window>
    </simple-container>

    <simple-container orientation="column"
      uniqueName="com.ibm.isclite.pagelayout.container.A2">
      <window uniqueName="com.ibm.isclite.pagelayout.window.A2">
        <title>
          <base:nls-string lang="en">Column Portlet</base:nls-string>
        </title>
      </window>
    </simple-container>
  </simple-container>
</layout-element>

```

Navigation element

Optional, and multiple are allowed. The `<navigation-element/>` element describes an element or *node* in the navigation tree. This element can be nested. Navigation elements also require unique names in the installation.

Navigation elements are arranged in a tree that reflects the structure of the navigation hierarchy that is created in the portal when the application is deployed. For example, a navigation tree with two primary nodes and two branch nodes each would exhibit the following structure:

```
<navigation-element>
  <navigation-element/>
  <navigation-element/>
</navigation-element>
<navigation-element>
  <navigation-element/>
  <navigation-element/>
</navigation-element>
```

In most cases, a navigation element references the layout element of the page that is displayed when the node is selected.

The following are the attributes of the `<navigation-element/>` element.

hidden

Optional. The default is `false`. When `true`, this attribute hides the navigation element from the console navigation tree. When this attribute hides a parent navigation element, it also hides any children. It is recommended that you associate a **parentTreeRef** attribute containing an empty string with each hidden navigation element. If you are planning to launch a navigation node, the navigation node must point to a layout element. In other words the navigation element must contain the **layoutElementRef** attribute.

uniqueName

Required. Specifies a unique identifier for the navigation element. To guarantee uniqueness, see “Console module elements - guidelines for unique identifiers” on page 111.

You can also use this attribute to associate a navigation element from another product with this product. In this case, you must omit the title element from the navigation element. For example, if Product A has a navigation element titled “Setting the background color” that also works with Product B, then Product B would include a navigation element without a title that specifies the same unique name as this node from Product A. In addition, you can specify a wild card pattern to include multiple navigation elements from another products. The wild card is specified using two dashes (`--`). For example, the following pattern would include all of the navigation elements from a product that is already deployed and that have a unique name starting with `com.ibm`.

```
<navigation-element uniqueName="com.ibm.--">
</navigation-element>
```

layoutElementRef

Optional. Specifies the page to be displayed when the node is selected.

The following elements make up the content of the navigation element.

<title/>

Optional, no more than one allowed. This element provides a title for the node to be displayed in the navigation. This title is also displayed in the task tree for My Tasks, and in the View selection menu over the navigation . Because of the limited viewable width of the View selection menu, try to keep the character length of the title to a minimum. To provide a localized string, use the `<base:nls-string/>` or `<base:nls-ref/>` elements for the title content.

<parent-tree/>

Optional, no more than one allowed. Specifies the parent node and position under the parent node

where the current navigation element is to be located. The parent node need not be part of the current application definition. If a requested parent node does not exist, the node is appended to the bottom of the navigation tree and is merged into the parent when the parent application is deployed.

The following attributes are used with this element.

parentTreeRef

Required. Indicates the unique name of the navigation element under which this node should be placed.

ordinal

Optional. Provides a non negative integer to determine the relative order of nodes under a particular parent tree. If no ordinal is specified, or if two navigation elements specify the same ordinal for a navigation level, the node is located in the order in which it is deployed.

The following example from the tree merge sample shows how the <parent-tree/> element is used to merge the navigation elements into the navigation for the page layout sample. The parentTreeRef attribute specifies the unique name of the top level navigation element for the page layout sample. Child navigation elements are placed under the corresponding top level node for tree merge. Ordinal is not specified, so this navigation tree is appended to the page layout navigation tree.

```
<navigation-element uniqueName="com.ibm.isclite.samples.navigationElement.TreeMerge">
    ...
    <parent-tree parentTreeRef="com.ibm.isclite.samples.navigationElement.Topology"/>
    <navigation-element uniqueName="com.ibm.isclite.treemerge.navigationElement.A"
        layoutElementRef="com.ibm.isclite.TreeMerge.layoutElement.A">
        ...
    </navigation-element>
    <navigation-element uniqueName="com.ibm.isclite.treemerge.navigationElement.B"
        layoutElementRef="com.ibm.isclite.TreeMerge.layoutElement.B">
        ...
    </navigation-element>
</navigation-element>
```

<navigation-content/>

Optional, multiple allowed. This element defines the content of a window in a page layout. This can be used to supply the content for a window at runtime or to map different portlets to the same window, depending on which navigation element is selected.

The following attributes are used with this element.

uniqueName

Required. Specifies a unique identifier for the navigation content. To guarantee uniqueness, see “Console module elements - guidelines for unique identifiers” on page 111.

componentDefinitionRef

Required. Indicates the unique name of the component definition that should be rendered in the window. The component definition can be a portlet entity or a url link. If the component definition specifies a url-link, omit the windowRef attribute. Windows cannot render external URLs.

windowRef

Optional. Indicates the unique name of the window where the component definition should be rendered. If this attribute is not specified, the component definition applies to the entire layout element. Multiple <navigation-content/> elements can specify the same window.
Optional.

The following example from the page layout sample illustrates how the <navigation-content/> element can be used.

1. A portlet is defined in a component definition with the unique name `com.ibm.isclite.pagelayout.appElement.B`.

```
<portlet-entity uniqueName="com.ibm.isclite.pagelayout.appElement.B"
                portletDefinitionRef="com.ibm.isclite.pagelayout.appElementDefinition.B">
  <title>
    <base:nls-string lang="en">ColumnPortlet</base:nls-string>
  </title>
</portlet-entity>
```

2. A window is defined without a component definition reference, which means it has empty content.

```
<simple-container orientation="column"
                 uniqueName="com.ibm.isclite.pagelayout.container.A2">
  <window uniqueName="com.ibm.isclite.pagelayout.window.A2">
    <title>
      <base:nls-string lang="en">Column Portlet</base:nls-string>
    </title>
  </window>
</simple-container>
```

3. The navigation content specifies the unique name of the portlet and window for the `componentDefinitionRef` and `windowRef` attributes respectively.

```
<!--This Navigation Element is for Layout A -->
<navigation-element uniqueName="com.ibm.isclite.pagelayout.navigationElement.pagelayoutA"
                   layoutElementRef="com.ibm.isclite.pagelayout.layoutElement.A">
  <title>
    <base:nls-string lang="en">2 Separate Containers</base:nls-string>
  </title>
  <navigation-content uniqueName="com.ibm.isclite.pagelayout.navigationContent.A2"
                    componentDefinitionRef="com.ibm.isclite.pagelayout.appElement.B"
                    windowRef="com.ibm.isclite.pagelayout.window.A2"/>
</navigation-element>
```

<preference/>

Optional. Indicates the unique name of the application definition for a product. This is used to include the node in the navigation tree of the product when console users select the product title to filter the navigation. For example:

1. The page layout sample is deployed with the title *Integrated Solutions Console Sample* (as defined by the title tag for the application definition). This title is added to the product filter of the navigation **View** selection list.
2. The tree merge sample is subsequently deployed specifying page layout as a prerequisite application. To ensure that navigation nodes for tree merge are also displayed when the console user selects *Integrated Solutions Console Sample* for product filtering, the navigation elements includes this preference definition.

```
<preference name="ProductFilter">
  <base:value>
    <base:string>com.ibm.isclite.samples.PageLayout</base:string>
  </base:value>
</preference>
```

Note: The name attribute must be set to `ProductFilter`. The `<base:string/>` element must indicate the unique name of the application definition for the prerequisite product.

Organizing the navigation tree: The console navigation tree provides access to the pages for which the user has permission. The hierarchy of nodes and pages on the navigation tree is defined by the `<navigation-element/>` element in the portal topology descriptor. All pages accessible from the navigation tree must be defined within the content hierarchy.

Before you start authoring the <navigation-element/> section of the `ibm-portal-topology.xml` file, take time to plan the entries that you want to be displayed on the console navigation tree. Identify the pages that will be accessible from the console navigation tree. If possible, reuse existing primary organizational nodes and any appropriate sub-nodes in your console navigation tree.

Console module elements - guidelines for unique identifiers:

A console module consists of XML descriptors that must provide element identifiers that are unique throughout the application. This topic provides guidelines for setting these identifiers to help ensure uniqueness. Some development tools, like WebSphere Application Server Toolkit, assign randomly-generated unique names for you. If you need to manage unique names yourself, follow these guidelines.

It is recommended that product or product suite teams define a naming convention for their console modules. For maximum effectiveness, the conventions should be defined before console modules are developed to allow time for refinement and avoid rework. The following convention provides one guideline that can help manage the unique names for an application.

`namespace + module_name + element name + identifier`

The identifier is necessary to distinguish different elements in the same descriptor. For example, the following unique names are used in the portal topology descriptor of the TreeMerge sample:

```
<component-tree           uniqueName="com.ibm.isclite.TreeMerge.appTree"           ...
<portlet-definition       uniqueName="com.ibm.isclite.TreeMerge.appElementDefinition.A"   ...
<portlet-entity          uniqueName="com.ibm.isclite.TreeMerge.appElement.A"     ...
<portlet-definition     uniqueName="com.ibm.isclite.TreeMerge.appElementDefinition.B"   ...
<portlet-entity         uniqueName="com.ibm.isclite.TreeMerge.appElement.B"     ...
<layout-element         uniqueName="com.ibm.isclite.TreeMerge.layoutElement.A"     ...
<simple-container        uniqueName="com.ibm.isclite.TreeMerge.container.A"     ...
<window                 uniqueName="com.ibm.isclite.TreeMerge.window.A">       ...
<layout-element         uniqueName="com.ibm.isclite.TreeMerge.layoutElement.B"   ...
<simple-container        uniqueName="com.ibm.isclite.TreeMerge.container.B"     ...
<window                 uniqueName="com.ibm.isclite.TreeMerge.window.B"     ...
<navigation-element     uniqueName="com.ibm.isclite.treemerge.navigationElement.A"   ...
<navigation-element     uniqueName="com.ibm.isclite.treemerge.navigationElement.B"   ...
<navigation-content     uniqueName="com.ibm.isclite.treemerge.navigationContent.B"   ...
```

Portal security schema

The portal security descriptor describes application roles for a portal application and maps those roles to actual users and groups of the console.

Description of the portal security elements

<ibm-portal-security/>

Root element of the portal security descriptor. This element contains the following element.

<application-role/>

Optional and multiple allowed. Specifies an application role for the portal application. An application role maps a set of permissions, or role type, to a specific resource defined in the application's topology descriptor. The following attribute is used with this element.

uniqueName

Required. Specifies the administrative role for this application. The role can be one of the existing roles that are provided by the application server or you can specify a new role for the application that is created during deployment.

- When you specify one of the application server roles, make sure you use all lower case (for example, *administrator*, *operator*, *monitor*, or *configurator*) rather than the initial capitalization that appears in the console interface (*Administrator*, *Operator*, *Monitor*, or *Configurator*). If there is a mismatch in upper or lower casing (*MOnitor* instead of *monitor*), then the role cannot be created.

- When you specify a new role for an application, use lowercase characters as a guideline.
- The administrative console also provides the *all authenticated portal users* virtual role. Resources with *all authenticated portal users* permission are available to all authenticated users.
- When you specify a new role, it inherits the access rights of the *monitor* role when it is created. A user with the new administrative role has access to the same resources as a user with the *monitor* role.
- When you specify a resource in the topology descriptor without protecting it with an administrative role or virtual role, it becomes unavailable to any role.

To guarantee uniqueness, see “Console module elements - guidelines for unique identifiers” on page 111.

The following element makes up the content of the application role.

<portal-role/>

Optional and multiple allowed. Associates a resource with a role type. The role type determines what actions a user can perform on that resource. The following attributes can be used with this element.

object-ref

Specifies the unique name of a resource in the topology descriptor. For this release, only portlet entities and navigation elements are supported.

role-type

Specifies the set of actions the user with this role type can perform on the resource. The following values are allowed for this release (case-sensitive):

- **User**

A user with this role type is allowed to view the resource and access help.

- **Privileged User**

A user with this role type is allowed to view the resource, edit preferences, and access help.

For this release, role-type has meaning only for portlet entities, however, role-type must be specified for both. You should specify `role-type="User"` for navigation elements. These role mappings cannot be changed through the console. Instead, you can update the descriptor in its extracted location on the server. You must delete and redeploy the application to cause the changes to take effect.

The console module samples provide examples of how to develop the elements of the portal security descriptor.

Using administrative programs (JMX)

This topic describes how to use Java application programming interfaces (APIs) to administer WebSphere Application Server and to manage your applications.

Before you begin

You can administer WebSphere Application Server and your applications through tools that come with the product or through programming with the Java APIs.

The wsadmin scripting tool, the administrative console, and the administrative command-line tools come with the product. These administrative tools provide most of the functions that you need to manage the product and the applications that run in WebSphere Application Server. You can use the command-line tools from automation scripts to control the servers. Scripts that are written for the wsadmin scripting tool offer a wide range of possible custom solutions that you can develop quickly.

Investigate these tools with the Java APIs to determine the best ways to administer WebSphere Application Server and your applications. For information on the Java APIs, view the application programming interfaces documentation.

- **No action required for WAS JMX APIs:** Each Java virtual machine (JVM) in WebSphere Application Server includes an embedded implementation of Java Management Extensions (JMX). In Application Server, Version 5, the JVMs contain an implementation of the JMX 1.0 specification. In Application Server, Version 6.0 and later, the JVMs contain an implementation of the JMX 1.2 specification. The JMX 1.0 implementation used in Version 5 is the TMX4J package that IBM Tivoli products supply. The JMX 1.2 specification used in Version 6.0 and later is the open source mx4j package. The JMX implementation change across the releases does not affect the behavior of the JMX MBeans in the Application Server. No Application Server administrative application programming interfaces (APIs) are altered due to the change from the JMX V1.0 specification to the JMX V1.2 specification.
- **Action might be required for custom MBeans:** The JMX V1.2 specification is compatible with the earlier JMX V1.0 specification. However, you might need to migrate custom MBeans that are supplied by products other than the Application Server from Version 5 to Version 6.0 and later. The primary concern for these custom MBeans is related to the values that are used in key properties of the JMX ObjectName class for the MBean. The open source mx4j implementation more stringently enforces property validation according to the JMX 1.2 specification. Test the custom MBeans that you deployed in Version 5 in Version 6.0 and later, to ensure compatibility. Full details of the JMX V1.2 specification changes from the JMX V1.0 specification are available in the JMX 1.2 specification.

About this task

WebSphere Application Server supports access to the administrative functions through a set of Java classes and methods. You can write a Java program that performs any of the administrative features of the WebSphere Application Server administrative tools. You can also extend the basic WebSphere Application Server administrative system to include your own managed resources.

You can prepare, install, uninstall, edit, and update applications through programming. Preparing an application for installation involves collecting various types of WebSphere Application Server-specific binding information to resolve references that are defined in the application deployment descriptors. This information can also be modified after installation by editing a deployed application. Updating consists of adding, removing or replacing a single file or a single module in an installed application, or supplying a partial application that manipulates an arbitrary set of files and modules in the deployed application. Updating the entire application uninstalls the old application and installs the new one. Uninstalling an application removes it entirely from the WebSphere Application Server configuration.

Perform any or all of the following tasks to manage WebSphere Application Server and your Java Platform, Enterprise Edition (Java EE) applications through programming.

- Create a JMX remote client program by using the JMX remote API (JSR 160)..
This topic describes how to develop a JMX remote program that uses the JMX remote API (JSR 160) to access the WebSphere Application Server administrative system.
- Create a custom Java administrative client program using the Java administrative APIs.
This topic describes how to develop a Java program that uses the WebSphere Application Server administrative APIs to access the administrative system of WebSphere Application Server.
- Extend the WebSphere Application Server administrative system with custom MBeans.

This topic describes how to extend the WebSphere Application Server administration system by supplying and registering new JMX MBeans in one of the Application Server processes. In this case, you can use the administrative classes and methods to add newly managed objects to the administrative system.

- Deploy and manage a custom Java administrative client program for use with multiple Java Platform, Enterprise Edition application servers.

This topic describes how to connect to a Java EE server, and how to manage multiple vendor servers.

Results

Depending on which tasks you complete, you have created your own administrative program, extended the WebSphere Application Server administrative console, connected and managed vendor servers, or managed your applications through programming.

What to do next

You can continue to administer WebSphere Application Server and your applications through programming or in combination with the tools that come with the WebSphere Application Server.

Java Management Extensions

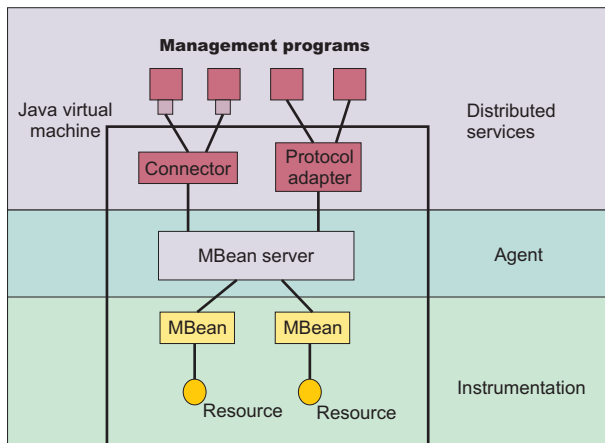
This topic gives an overview of Java Management Extensions (JMX) in general and how this standard applies to WebSphere Application Server.

Java Management Extensions overview

Java Management Extensions (JMX) is the Java standard for managing application resources. The management architecture that is defined by JMX is divided into three levels:

- The bottom level is management instrumentation. Each manageable resource is described by an interface that specifies the attributes it has, the operations it supports, and the notifications it sends. This resource is a managed bean (MBean).
- The middle level is the management agent. Each managed process contains a JMX agent that includes an MBean server, which provides a registry and access point for MBeans. Management clients must use the MBean server to access the registered MBeans.
- The top level of the architecture is defined by JMX Remote application programming interface (API) (JSR 160). JSR 160 uses Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP), but is not interoperable with the RMI connector. The RMI, SOAP/HTTP, and SOAP/HTTPS connectors were created before the JSR160 specification and are supported. The Inter-Process Communications (IPC) connector is also supported.

The top level of the architecture is the distributed services level, and its role is to facilitate remote access to JMX agents. This task is accomplished through connectors, which provide a protocol-independent, location-transparent, client-side interface to the MBean server (for example, a Remote Method Invocation (RMI) connector), or protocol adapters, which provide protocol-specific, server-side access to the MBean server (for example, an HTTP adapter).

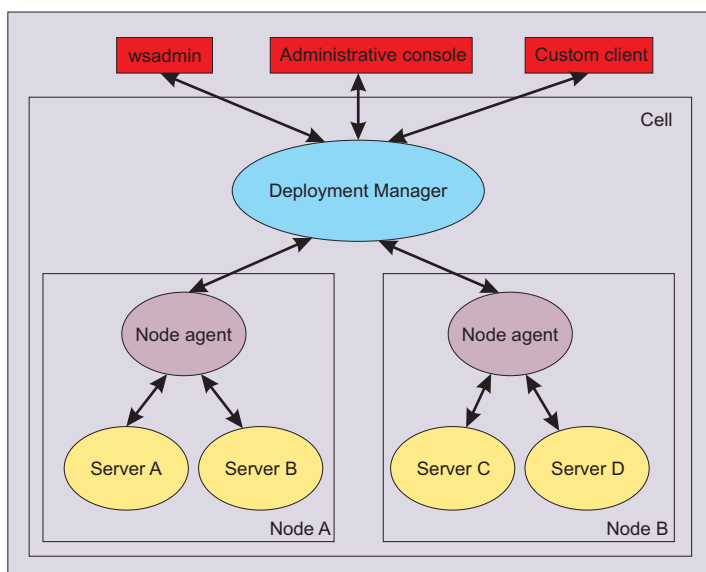


Java Management Extensions in WebSphere Application Server

Java Management Extensions (JMX) is at the core of Application Server administration capabilities. The application server contains a JMX agent. All of the system components are defined as MBeans. The JMX agent in Application Server supports the following connectors: JSR160RMI, Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOP), Simple Object Access Protocol/Hypertext Transfer Protocol (SOAP/HTTP), Simple Object Access Protocol/Hypertext Transfer Protocol Secure (SOAP/HTTPS), and Inter-Process Communications (IPC), which provides remote access to the server resources. All of the administration tools included with Application Server use these JMX facilities to accomplish their functions.

In a stand-alone Application Server installation, servers exist and are administered individually. An administrative client connects directly to the Application Server in this environment.

Application Server provides an AdminService class that reflects the standard JMX MBeanServer interface, and wraps the MBeanServer interface so that it takes part in implementing this distributed management functionality.



Creating a custom Java administrative client program using WebSphere Application Server administrative Java APIs

This section describes how to develop a Java program for accessing the WebSphere Application Server administrative system by using the WebSphere Application Server administrative application programming interfaces (APIs).

Before you begin

This task assumes a basic familiarity with Java Management Extensions (JMX) API programming. For information on the Java APIs, view the application programming interfaces documentation.

About this task

When you develop and run administrative clients that use various JMX connectors and that have security enabled, use the following guidelines. When you follow these guidelines, you guarantee the behavior among different implementations of JMX connectors. Any programming model that strays from these guidelines is unsupported.

1. Create and use a single administrative client before you create and use another administrative client.
2. Create and use an administrative client on the same thread.
3. Use one of the following ways to specify a user ID and password to create a new administrative client:
 - Specify a default user ID and password in the property file.
 - Specify a user ID and password other than the default. Once you create an administrative client with a nondefault user ID and password, specify the nondefault user ID and password when you create subsequent administrative clients.

1. Develop an administrative client program.
2. Build and run the administrative client program.

The steps required to build and run your program depends on the kind of application environment your code runs.

Refer to the Using application clients topic in the *Developing and deploying applications* PDF for details on how to build and run your administrative client program.

Developing an administrative client program

This topic describes how to develop an administrative client program that utilizes WebSphere Application Server administrative application programming interfaces (APIs) and Java Management Extensions (JMX).

About this task

WebSphere Application Server administrative APIs provide control of the operational aspects of your distributed system as well as the ability to update your configuration. For information about the AdminClient interface, view the application programming interfaces documentation.

This topic also demonstrates examples of MBean operations. For information about MBean programming, refer to the Reference materials. View the API documentation for MBean interfaces.

This example shows how to get and use a NodeAgent MBean, which is not available for your product. However, the description of how to get and put the NodeAgent MBean is similar to what you would do for other MBeans that are available for your product.

1. Create an AdminClient instance.

An administrative client program needs to invoke methods on the AdminService object that is running in the application server in the base installation. The AdminClient class provides a proxy to the remote AdminService object through one of the supported Java Management Extensions (JMX) connectors. The following example shows how to create an AdminClient instance:


```

Properties connectProps = new Properties();
connectProps.setProperty(
AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);

connectProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
connectProps.setProperty(AdminClient.CONNECTOR_PORT, "8879");
connectProps.setProperty(AdminClient.USERNAME, "test2");
connectProps.setProperty(AdminClient.PASSWORD, "user24test");
AdminClient adminClient = null;
try
{
    adminClient = AdminClientFactory.createAdminClient(connectProps);
}
catch (ConnectorException e)
{
    System.out.println("Exception creating admin client: " + e);
}

```

a. Set up a Properties object.

The example sets up a Properties object with the properties that are required to get to your server. In this case, you use the Simple Object Access Protocol (SOAP) connector to reach the server; for the connector type, use the value: AdminClient.CONNECTOR_TYPE_SOAP.

b. For simplicity, run the client program on the same machine as the server; use localhost for the host name.

To access a remote host instead of a local host, use a network resolvable name for that host.

c. Set the port number on which the server SOAP connector is listening.

In a single server installation, the default port number for the application server SOAP connector is 8880.

d. After the connection properties are set, use the AdminClientFactory class and the Properties object to create an AdminClient object that is connected to your chosen server.

Depending on factors such as your desired protocol and security environment, you might need to set other properties. For example, if you enable security for your application client program, include the javax.net.ssl.* properties. For more detailed information about the AdminClient interface, the javax.net.ssl.* properties, and additional creation examples, refer to the AdminClient interface in the application programming interfaces documentation.

2. Find an MBean. When you obtain an AdminClient instance, you can use it to access managed resources in the administration servers and application servers. Each managed resource registers an MBean with the AdminService through which you can access the resource. The MBean is represented by an ObjectName instance that identifies the MBean. An ObjectName instance consists of a domain name followed by an unordered set of one or more key properties. The syntax for the domain name follows:

```
[domainName]:property=value[,property=value]*
```

For WebSphere Application Server, the domain name is WebSphere and the key properties defined for administration are as follows:

type	The type of MBean. For example: Server, TraceService, Java virtual machine (JVM).
name	The name identifier for the individual instance of the MBean.
cell	The name of the cell that the MBean is running.
node	The name of the node that the MBean is running.
process	The name of the process that the MBean is running.

Some MBeans in WebSphere Application Server use additional key properties. An MBean without key properties can be registered with the MBean server in a WebSphere Application Server process.

However, such an MBean cannot participate in the distributed enhancements that WebSphere Application Server adds, for example, request routing, distributed event notification, and so on.

If you know the complete set of key properties for an ObjectName instance, you can use it to find the MBean it identifies. However, finding MBeans without having to know all of their key properties is usually more practical and convenient. Use the wildcard character asterisk (*) for any key properties that you do not need to match. The following table provides some examples of object names with wildcard key properties that match single or multiple MBeans.

:type=Server,	All MBeans of type Server
:node=Node1,type=Server,	All MBeans of type Server on Node1
:type=JVM,process=server1,node=Node1,	The JVM MBean in the server named server1 node Node1
:process=server1,	All MBeans in all servers named server1
:process=server1,node=Node1,	All MBeans in the server named server1 on Node1

You can locate an MBean by querying for it with object names that match key properties. The following example shows how to find the MBean for the node agent of node, MyNode:

```
String nodeName = "MyNode";
String query = "WebSphere:type=NodeAgent,node=" + nodeName + ",*";
ObjectName queryName = new ObjectName(query);
ObjectName nodeAgent = null;
Set s = adminClient.queryNames(queryName, null);
if (!s.isEmpty())
    nodeAgent = (ObjectName)s.iterator().next();
else
    System.out.println("Node agent MBean was not found");
```

- a. Build an ObjectName instance with a query string that specifies the key properties of type and node.

By using a wildcard for the remaining key properties, this pattern matches the object names for all MBeans of the type NodeAgent on the node MyNode. Because only one node agent per node exists, this information is sufficient to identify the MBean that you want.

- b. Give this ObjectName instance to the queryNames method of the AdminClient interface.

The AdminClient interface performs the remote call to the AdminService interface to obtain the set of MBean object names that match the query. The null second parameter to this method is a query expression (QueryExp) object that you can use as an additional query over the MBeans that match the ObjectName pattern in the first parameter.

- c. Use the set iterator to get the first and, in this case, only element.

The element is the MBean ObjectName instance of the node agent.

3. Use the MBean. What a particular MBean can do depends on the management interface of that MBean. An MBean can declare:

- Attributes that you can obtain or set
- Operations that you can invoke
- Notifications for which you can register listeners
-

For the MBeans provided by WebSphere Application Server, you can find information about the interfaces they support in the MBean API documentation. The following example invokes one of the operations available on the NodeAgent MBean that you located previously. The following example starts the *MyServer* application server:

```
String opName = "launchProcess";
String signature[] = { "java.lang.String" };
String params[] = { "MyServer" };
try
{
    adminClient.invoke(nodeAgent, opName, params, signature);
}
```

```

}
catch (Exception e)
{
    System.out.println("Exception invoking launchProcess: " + e);
}

```

The `AdminClient.invoke` method is a generic means of invoking any operation on any MBean. The parameters are:

- The object name of the target MBean, `nodeAgent`
- The name of the operation, `opName`
- An object array that contains the operation parameters, `params`
- A string array that contains the operation signature, `signature`

The `launchProcess` operation in the example has a single parameter which is a string that identifies the server to start.

The `invoke` method returns an object instance, which the calling code can use to cast to the correct return type for the invoked operation. The `launchProcess` operation is declared `void` so that you can ignore the return value in this example.

4. Register for events. In addition to managing resources, the JMX API also supports application monitoring for specific administrative events. Certain events produce notifications, for example, when a server starts. Administrative applications can register as listeners for these notifications. The WebSphere Application Server provides a full implementation of the JMX notification model, and provides additional function so you can receive notifications in a distributed environment. For a complete list of the notifications emitted from WebSphere Application Server MBeans, refer to the `com.ibm.websphere.management.NotificationConstants` class in the MBean API documentation. The following example shows how an object can register for event notifications that are emitted from an MBean using the `ObjectName` node agent:

```
adminClient.addNotificationListener(nodeAgent, this, null, null);
```

In this example, the first parameter is the `ObjectName` for the node agent MBean. The second parameter identifies the listener object, which must implement the `NotificationListener` interface. In this case, the calling object is the listener. The third parameter is a filter that you can use to indicate which notifications you want to receive. When you leave this value as `null`, you receive all notifications from this MBean. The final parameter is a handback object that you can use to set the JMX API to return to you when it emits a notification.

Another enhanced feature that Application Server provides is the ability to register as a notification listener of multiple MBeans with one call. This registration is done through the `addNotificationListenerExtended` method of the `AdminClient` interface, an extension of the standard JMX `addNotificationListener` method. This extension method even lets you register for MBeans that are not currently active. This registration is important in situations where you want to monitor events from resources that can be stopped and restarted during the lifetime of your administrative client program.

5. Handle the events. Objects receive JMX event notifications through the `handleNotification` method, which is defined by the `NotificationListener` interface and which any event receiver must implement. The following example is an implementation of the `handleNotification` method that reports the notifications that it receives:

```

public void handleNotification(Notification n, Object handback)
{
    System.out.println("*****");
    System.out.println("* Notification received at " + new Date().toString());
    System.out.println("* type      = " + ntfyObj.getType());
    System.out.println("* message = " + ntfyObj.getMessage());
    System.out.println("* source  = " + ntfyObj.getSource());
    System.out.println(
        "* seqNum  = " + Long.toString(ntfyObj.getSequenceNumber());
    System.out.println("* timeStamp = " + new Date(ntfyObj.getTimeStamp()));
    System.out.println("* userData = " + ntfyObj.getUserData());
    System.out.println("*****");
}

```

Example: Administrative client program

This example shows how to connect to the node agent server, which is not available for your product. However, you can connect to your server by changing the host and port values. Substitute your server for the node agent server references. Since the NodeAgent MBean is not available for your product, substitute the queryNames string to search for another MBean.

Copy the contents to a file named AdminClientExample.java. After changing the node name and server name to the appropriate values for your configuration, you can compile and run it using the instructions from [Creating a custom Java administrative client program using WebSphere Application Server administrative Java APIs](#)

```
import java.util.Date;
import java.util.Properties;
import java.util.Set;

import javax.management.InstanceNotFoundException;
import javax.management.MalformedObjectNameException;
import javax.management.Notification;
import javax.management.NotificationListener;
import javax.management.ObjectName;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;

public class AdminClientExample implements NotificationListener
{
    private AdminClient adminClient;
    private ObjectName nodeAgent;
    private long notifyCount = 0;

    public static void main(String[] args)
    {
        AdminClientExample ace = new AdminClientExample();

        // Create an AdminClient
        ace.createAdminClient();

        // Find a NodeAgent MBean
        ace.getNodeAgentMBean("ellington");

        // Invoke launchProcess
        ace.invokeLaunchProcess("server1");

        // Register for NodeAgent events
        ace.registerNotificationListener();

        // Run until interrupted
        ace.countNotifications();
    }

    private void createAdminClient()
    {
        // Set up a Properties object for the JMX connector attributes
        Properties connectProps = new Properties();
        connectProps.setProperty(
            AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
        connectProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
        connectProps.setProperty(AdminClient.CONNECTOR_PORT, "8879");

        // Get an AdminClient based on the connector properties
        try
        {
            adminClient = AdminClientFactory.createAdminClient(connectProps);
        }
    }
}
```

```

    }
    catch (ConnectorException e)
    {
        System.out.println("Exception creating admin client: " + e);
        System.exit(-1);
    }
}
System.out.println("Connected to DeploymentManager");
}

private void getNodeAgentMBean(String nodeName)
{
    // Query for the ObjectName of the NodeAgent MBean on the given node
    try
    {
        String query = "WebSphere:type=NodeAgent,node=" + nodeName + ",*";
        ObjectName queryName = new ObjectName(query);
        Set s = adminClient.queryNames(queryName, null);
        if (!s.isEmpty())
            nodeAgent = (ObjectName)s.iterator().next();
        else
        {
            System.out.println("Node agent MBean was not found");
            System.exit(-1);
        }
    }
    catch (MalformedObjectNameException e)
    {
        System.out.println(e);
        System.exit(-1);
    }
    catch (ConnectorException e)
    {
        System.out.println(e);
        System.exit(-1);
    }

    System.out.println("Found NodeAgent MBean for node " + nodeName);
}

private void invokeLaunchProcess(String serverName)
{
    // Use the launchProcess operation on the NodeAgent MBean to start
    // the given server
    String opName = "launchProcess";
    String signature[] = { "java.lang.String" };
    String params[] = { serverName };
    boolean launched = false;
    try
    {
        Boolean b = (Boolean)adminClient.invoke(
nodeAgent, opName, params, signature);
        launched = b.booleanValue();
        if (launched)
            System.out.println(serverName + " was launched");
        else
            System.out.println(serverName + " was not launched");
    }
    catch (Exception e)
    {
        System.out.println("Exception invoking launchProcess: " + e);
    }
}
}

```

```

private void registerNotificationListener()
{
    // Register this object as a listener for notifications from the
    // NodeAgent MBean. Don't use a filter and don't use a handback
    // object.
    try
    {
        adminClient.addNotificationListener(nodeAgent, this, null, null);
        System.out.println("Registered for event notifications");
    }
    catch (InstanceNotFoundException e)
    {
        System.out.println(e);
        e.printStackTrace();
    }
    catch (ConnectorException e)
    {
        System.out.println(e);
        e.printStackTrace();
    }
}

public void handleNotification(Notification ntfyObj, Object handback)
{
    // Each notification that the NodeAgent MBean generates will result in
    // this method being called
    ntfyCount++;
    System.out.println("*****");
    System.out.println("* Notification received at " + new Date().toString());
    System.out.println("* type          = " + ntfyObj.getType());
    System.out.println("* message       = " + ntfyObj.getMessage());
    System.out.println("* source        = " + ntfyObj.getSource());
    System.out.println(
        "* seqNum      = " + Long.toString(ntfyObj.getSequenceNumber());
    System.out.println("* timeStamp = " + new Date(ntfyObj.getTimeStamp());
    System.out.println("* userData  = " + ntfyObj.getUserData());
    System.out.println("*****");
}

private void countNotifications()
{
    // Run until killed
    try
    {
        while (true)
        {
            Thread.currentThread().sleep(60000);
            System.out.println(ntfyCount + " notification have been received");
        }
    }
    catch (InterruptedException e)
    {
    }
}
}

```

Example: Administrative client program:

This example is a complete administrative client program.

This example shows how to connect to the node agent server, which is not available for your product. However, you can connect to your server by changing the host and port values. Substitute your server for the node agent server references. Since the NodeAgent MBean is not available for your product, substitute the queryNames string to search for another MBean.

Copy the contents to a file named AdminClientExample.java. After changing the node name and server name to the appropriate values for your configuration, you can compile and run it using the instructions from [Creating a custom Java administrative client program using WebSphere Application Server administrative Java APIs](#)

```
import java.util.Date;
import java.util.Properties;
import java.util.Set;

import javax.management.InstanceNotFoundException;
import javax.management.MalformedObjectNameException;
import javax.management.Notification;
import javax.management.NotificationListener;
import javax.management.ObjectName;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;

public class AdminClientExample implements NotificationListener
{
    private AdminClient adminClient;
    private ObjectName nodeAgent;
    private long notifyCount = 0;

    public static void main(String[] args)
    {
        AdminClientExample ace = new AdminClientExample();

        // Create an AdminClient
        ace.createAdminClient();

        // Find a NodeAgent MBean
        ace.getNodeAgentMBean("ellington");

        // Invoke launchProcess
        ace.invokeLaunchProcess("server1");

        // Register for NodeAgent events
        ace.registerNotificationListener();

        // Run until interrupted
        ace.countNotifications();
    }

    private void createAdminClient()
    {
        // Set up a Properties object for the JMX connector attributes
        Properties connectProps = new Properties();
        connectProps.setProperty(
            AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
        connectProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
        connectProps.setProperty(AdminClient.CONNECTOR_PORT, "8879");

        // Get an AdminClient based on the connector properties
        try
        {
            adminClient = AdminClientFactory.createAdminClient(connectProps);
        }
        catch (ConnectorException e)
        {
        }
    }
}
```

```

        {
            System.out.println("Exception creating admin client: " + e);
            System.exit(-1);
        }

        System.out.println("Connected to DeploymentManager");
    }

private void getNodeAgentMBean(String nodeName)
{
    // Query for the ObjectName of the NodeAgent MBean on the given node
    try
    {
        String query = "WebSphere:type=NodeAgent,node=" + nodeName + ",*";
        ObjectName queryName = new ObjectName(query);
        Set s = adminClient.queryNames(queryName, null);
        if (!s.isEmpty())
            nodeAgent = (ObjectName)s.iterator().next();
        else
        {
            System.out.println("Node agent MBean was not found");
            System.exit(-1);
        }
    }
    catch (MalformedObjectNameException e)
    {
        System.out.println(e);
        System.exit(-1);
    }
    catch (ConnectorException e)
    {
        System.out.println(e);
        System.exit(-1);
    }

    System.out.println("Found NodeAgent MBean for node " + nodeName);
}

private void invokeLaunchProcess(String serverName)
{
    // Use the launchProcess operation on the NodeAgent MBean to start
    // the given server
    String opName = "launchProcess";
    String signature[] = { "java.lang.String" };
    String params[] = { serverName };
    boolean launched = false;
    try
    {
        Boolean b = (Boolean)adminClient.invoke(
nodeAgent, opName, params, signature);
        launched = b.booleanValue();
        if (launched)
            System.out.println(serverName + " was launched");
        else
            System.out.println(serverName + " was not launched");
    }
    catch (Exception e)
    {
        System.out.println("Exception invoking launchProcess: " + e);
    }
}

private void registerNotificationListener()
{

```



```

// Register this object as a listener for notifications from the
// NodeAgent MBean. Don't use a filter and don't use a handback
// object.
try
{
    adminClient.addNotificationListener(nodeAgent, this, null, null);
    System.out.println("Registered for event notifications");
}
catch (InstanceNotFoundException e)
{
    System.out.println(e);
}
catch (ConnectorException e)
{
    System.out.println(e);
}
}

public void handleNotification(Notification ntfyObj, Object handback)
{
    // Each notification that the NodeAgent MBean generates will result in
    // this method being called
    ntfyCount++;
    System.out.println("*****");
    System.out.println("* Notification received at " + new Date().toString());
    System.out.println("* type      = " + ntfyObj.getType());
    System.out.println("* message   = " + ntfyObj.getMessage());
    System.out.println("* source    = " + ntfyObj.getSource());
    System.out.println(
        "* seqNum    = " + Long.toString(ntfyObj.getSequenceNumber());
    System.out.println("* timeStamp = " + new Date(ntfyObj.getTimeStamp()));
    System.out.println("* userData  = " + ntfyObj.getUserData());
    System.out.println("*****");
}

private void countNotifications()
{
    // Run until killed
    try
    {
        while (true)
        {
            Thread.currentThread().sleep(60000);
            System.out.println(ntfyCount + " notification have been received");
        }
    }
    catch (InterruptedException e)
    {
    }
}
}

```

Creating a Java Management Extensions client program using the Java Management Extensions Remote application programming interface

This topic describes how to develop and build a Java Management Extensions (JMX) client program that is compliant with JMX Remote application programming interface (JSR 160). After you have a working JMX client program, you can use it to manage WebSphere Application Server or non-WebSphere Application Server systems.

Before you begin

This task assumes a basic familiarity with JSR 160 and JMX application programming interface (API) programming. For information on JSR 160, see <http://www.jcp.org/en/jsr/detail?id=160>. For information on the Java APIs, view the application programming interfaces documentation.

About this task

When you develop and run JMX clients that use various JMX connectors and that have security enabled, use the following guidelines. When you follow these guidelines, you guarantee the behavior among different implementations of JMX connectors. Any programming model that strays from these guidelines is unsupported.

1. Create and use a single JMX client before you create and use another JMX client.
2. Create and use a JMX client on the same thread.
3. Use one of the following ways to specify a user ID and password to create a new JMX client:
 - Specify a default user ID and password in the property file.
 - Specify a user ID and password other than the default. After you create a JMX client with a nondefault user ID and password, specify the nondefault user ID and password when you create subsequent JMX clients.
1. Develop a JMX client program.
2. Build and run the JMX client program.

The steps that are required to build and run your program depend on the kind of application environment that your code runs. Refer to the Using application clients topic in the *Developing and deploying applications* PDF for details on how to build and run your JMX client program.

Results

You have developed, built, and run a JMX client program that is JSR 160 compliant.

Developing a Java Management Extensions client program using Java Management Extensions Remote application programming interface

This topic describes how to develop a Java Management Extensions (JMX) connector specification and JMX Remote application programming interface (API) (JSR 160). The program can communicate by Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP)

Before you begin

This topic assumes a basic understanding of JSR 160, JMX APIs, and managed beans (MBeans). For more information on JSR 160, see <http://www.jcp.org/en/jsr/detail?id=160>. For more information on the JMX APIs and on MBeans, view the application programming interfaces documentation.

About this task

You can administer your WebSphere Application Server environment through the administrative console, the wsadmin utility, or Java Management Extensions (JMX) programming. This topic discusses how to develop a JMX remote client program using the JMX remote API so that you can administer your WebSphere Application Server environment through JMX programming.

1. Specify the JMX connector address for the server through the JMXServiceURL class.
The form of the JMXServiceURL class is `service:jmx:rmi://host name:port/jndi/JMXConnector`
2. Specify the user ID and password for the server, if security is enabled.
3. Establish the JMX connection.
4. Get the MBean server connection instance.

Results

You have established a connection to WebSphere Application Server through an RMI connection and started the WebSphere Application Server through the node agent.

Example

The following example is for a thin application client.

```
import java.util.Date;
import java.util.Set;
import java.util.Hashtable;

import javax.management.InstanceNotFoundException;
import javax.management.MalformedObjectNameException;
import javax.management.Notification;
import javax.management.NotificationListener;
import javax.management.ObjectName;
import javax.management.MBeanServerConnection;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;

public class JMXRemoteClientApp implements NotificationListener {

    private MBeanServerConnection mbsc = null;
    private ObjectName nodeAgent;
    private long ntfyCount = 0;

    public static void main(String[] args)
    {
        try {

            JMXRemoteClientApp client = new JMXRemoteClientApp();

            String hostname=args[0];
            String port=args[1];
            String nodeName =args[2];

            client.connect(hostname,port);

            // Find a node agent MBean
            client.getNodeAgentMBean(nodeName);

            // Invoke the launch process.
            client.invokeLaunchProcess("server1");

            // Register for node agent events
            client.registerNotificationListener();

            // Run until interrupted.
            client.countNotifications();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void connect(String hostname,String port) throws Exception
    {
        String jndiPath="/jndi/JMXConnector";

        JMXServiceURL url = new JMXServiceURL("service:jmx:iiop://" +hostname+" "+port+jndiPath);
```

```

Hashtable h = new Hashtable();

//Specify the user ID and password for the server if security is enabled on server.

// String[] credentials = new String[] {username ,password };
//h.put("jmx.remote.credentials", credentials);

//Establish the JMX connection.

JMXConnector jmx = JMXConnectorFactory.connect(url, h);

//Get the MBean server connection instance.

mbsc = jmx.getMBeanServerConnection();

System.out.println("Connected to DeploymentManager");
}

private void getNodeAgentMBean(String nodeName)
{
    // Query for the object name of the node agent MBean on the given node
    try {
        String query = "WebSphere:type=NodeAgent,node=" + nodeName + ",*";
        ObjectName queryName = new ObjectName(query);
        Set s = mbsc.queryNames(queryName, null);
        if (!s.isEmpty())
            nodeAgent = (ObjectName)s.iterator().next();
        else {
            System.out.println("Node agent MBean was not found");
            System.exit(-1);
        }
    } catch (Exception e) {
        System.out.println(e);
        System.exit(-1);
    }

    System.out.println("Found NodeAgent MBean for node " + nodeName);
}

private void invokeLaunchProcess(String serverName)
{
    // Use the launch process on the node agent MBean to start
    // the given server.
    String opName = "launchProcess";
    String signature[] = { "java.lang.String" };
    String params[] = { serverName };
    boolean launched = false;
    try {
        Boolean b = (Boolean)mbsc.invoke(nodeAgent, opName, params, signature);
        launched = b.booleanValue();
        if (launched)
            System.out.println(serverName + " was launched");
        else
            System.out.println(serverName + " was not launched");
    } catch (Exception e) {
        System.out.println("Exception invoking launchProcess: " + e);
    }
}

private void registerNotificationListener()
{
    // Register this object as a listener for notifications from the
    // node agent MBean. Do not use a filter and do not use a handback
    // object.

```

```

    try {
        mbsc.addNotificationListener(nodeAgent, this, null, null);
        System.out.println("Registered for event notifications");
    } catch (Exception e) {
        System.out.println(e);
    }
}

public void handleNotification(Notification ntfyObj, Object handback)
{
    // Each notification that the node agent MBean generates results in
    // a call to this method.
    ntfyCount++;
    System.out.println("*****");
    System.out.println("* Notification received at " + new Date().toString());
    System.out.println("* type      = " + ntfyObj.getType());
    System.out.println("* message = " + ntfyObj.getMessage());
    System.out.println("* source  = " + ntfyObj.getSource());
    System.out.println(
        "* seqNum    = " + Long.toString(ntfyObj.getSequenceNumber()));
    System.out.println("* timeStamp = " + new Date(ntfyObj.getTimeStamp()));
    System.out.println("* userData  = " + ntfyObj.getUserData());
    System.out.println("*****");
}

private void countNotifications()
{
    // Run until stopped.
    try {
        while (true) {
            Thread.currentThread().sleep(60000);
            System.out.println(ntfyCount + " notification have been received");
        }
    } catch (InterruptedException e) {
    }
}
}

```

Extending the WebSphere Application Server administrative system with custom MBeans

You can extend the WebSphere Application Server administration system by supplying and registering new Java Management Extensions (JMX) MBeans (see JMX 1.x Specification for details) in one of the WebSphere processes.

About this task

JMX MBeans represent the management interface for a particular piece of logic. All of the managed resources within the standard WebSphere infrastructure are represented as JMX MBeans. There are a variety of ways in which you can create your own MBeans and register them with the JMX MBeanServer running in any WebSphere process. For more information, view the MBean API documentation.

1. Create custom JMX MBeans.

You have some alternatives to select from, when creating MBeans to extend the WebSphere administrative system. You can use any existing JMX MBean from another application. You can register any MBean that you tested in a JMX MBean server outside of the WebSphere Application Server environment in a WebSphere Application Server process, including standard MBeans, dynamic MBeans, open MBeans, and model MBeans.

In addition to any existing JMX MBeans, and ones that were written and tested outside of the WebSphere Application Server environment, you can use the special distributed extensions provided by WebSphere and create a WebSphere ExtensionMBean provider. This alternative provides better

integration with all of the distributed functions of the WebSphere administrative system. An ExtensionMBean provider implies that you supply an XML file that contains an MBean Descriptor based on the DTD shipped with the WebSphere Application Server. This descriptor tells the WebSphere system all of the attributes, operations, and notifications that your MBean supports. With this information, the WebSphere system can route remote requests to your MBean and register remote Listeners to receive your MBean event notifications.

All of the internal WebSphere MBeans follow the Model MBean pattern. Pure Java classes supply the real logic for management functions, and the WebSphere MBeanFactory class reads the description of these functions from the XML MBean Descriptor and creates an instance of a ModelMBean that matches the descriptor. This ModelMBean instance is bound to your Java classes and registered with the MBeanServer running in the same process as your classes. Your Java code now becomes callable from any WebSphere Application Server administrative client through the ModelMBean created and registered to represent it.

2. Optionally define an explicit MBean security policy.

If you do not define an MBean security policy, WebSphere Application Server uses the default security policy.

3. Register the new MBeans. There are various ways to register your MBean.

You can register your MBean with the WebSphere Application Server administrative service.

You can register your MBean with the MBeanServer in a WebSphere Application Server process. The following list describes the available options in order of preference:

- Go through the MBeanFactory class. If you want the greatest possible integration with the WebSphere Application Server system, then use the MBeanFactory class to manage the life cycle of your MBean through the activateMBean and deactivateMBean methods of the MBeanFactory class. Use these methods, by supplying a subclass of the RuntimeCollaborator abstract superclass and an XML MBean descriptor file. Using this approach, you supply a pure Java class that implements the management interface defined in the MBean descriptor. The MBeanFactory class creates the actual ModelMBean and registers it with the WebSphere Application Server administrative system on your behalf.

This option is recommended for registering model MBeans.

- Use the JMXManageable and CustomService interface. You can make the process of integrating with WebSphere administration even easier by implementing a CustomService interface that also implements the JMXManageable interface. Using this approach, you can avoid supplying the RuntimeCollaborator. When your CustomService interface is initialized, the WebSphere MBeanFactory class reads your XML MBean descriptor file and creates, binds, and registers an MBean to your CustomService interface automatically. After the shutdown method of your CustomService is called, the WebSphere Application Server system automatically deactivates your MBean.
- Go through the AdminService interface. You can call the registerMBean() method on the AdminService interface and the invocation is delegated to the underlying MBeanServer for the process, after appropriate security checks. You can obtain a reference to the AdminService using the getAdminService() method of the AdminServiceFactory class.

This option is recommended for registering standard, dynamic, and open MBeans. Implement the UserCollaborator class to use the MBeans and to provide a consistent level of support for them across distributed and z/OS platforms.

- Get MBeanServer instances directly. You can get a direct reference to the JMX MBeanServer instance running in any WebSphere Application Server process, by calling the getMBeanServer() method of the MBeanFactory class. You get a reference to the MBeanFactory class by calling the getMBeanFactory() method of the AdminService interface.

When a custom MBean is registered directly with the MBean server, the MBean object name is enhanced with the cell, node and process name keys by default. This registration allows the MBean to participate in the distributed features of the WebSphere Application Server administrative system. You can turn off the default behavior by setting the com.ibm.websphere.mbeans.disableRouting custom property.

See the *Installing your application serving environment* PDF for more information on the `com.ibm.websphere.mbeans.disableRouting` custom property.

Results

Regardless of the approach used to create and register your MBean, you must set up proper Java 2 security permissions for your new MBean code. The WebSphere AdminService and MBeanServer are tightly protected using Java 2 security permissions and if you do not explicitly grant your code base permissions, security exceptions are thrown when you attempt to invoke methods of these classes. If you are supplying your MBean as part of your application, you can set the permissions in the `was.policy` file that you supply as part of your application metadata. If you are using a CustomService interface or other code that is not delivered as an application, you can edit the `library.policy` file in the node configuration, or even the `server.policy` file in the `properties` directory for a specific installation.

Best practices for standard, dynamic, and open MBeans

This article discusses recommended guidelines for standard, dynamic, and open MBeans.

The underlying interface for the WebSphere Application Server administrative service is AdminService. Remote access occurs through the AdminControl scripting object.

For WebSphere Application Server Version 5, the MBean registration and capabilities are as follows:

MBean type	Registered with:	Capabilities
Model	WebSphere Application Server administrative service	Local access is through the WebSphere Application Server administrative service or the MBean server. Remote access is through the WebSphere Application Server administrative service, and WebSphere Application Server security.
Standard, dynamic, or open	MBean server	Local access is through the WebSphere Application Server administrative service or the MBean server on the distributed platform.

Note: For Version 6, you can optionally register standard, dynamic, and open custom MBeans with the WebSphere Application Server administrative service to take advantage of the capabilities that in Version 5 are available only to model MBeans.

Version 6 introduces a special run-time collaborator that you use with standard, dynamic or open custom MBeans to register the custom MBeans with the WebSphere Application Server administrative service. The standard, dynamic, and open MBeans display in the administrative service as model MBeans. The administrative service uses the capabilities available to MBeans that are registered with the administrative service.

For WebSphere Application Server Version 6, the MBean registration and capabilities are as follows:

MBean type	Registered with:	Capabilities
------------	------------------	--------------

Model, and optionally standard, dynamic, or open	WebSphere Application Server administrative service	Local access is through the WebSphere Application Server administrative service or the MBean server. Remote access is through the WebSphere Application Server administrative service, and WebSphere Application Server security.
Standard, dynamic, or open	MBean server	Local access is through the WebSphere Application Server administrative service or the MBean server on the distributed platform. Remote access is through the WebSphere Application Server administrative service, the Java Management Extensions (JMX) Remote application programming interface (API) (JSR 160) client code, and WebSphere Application Server security.

Creating and registering standard, dynamic, and open custom MBeans

You can create standard, dynamic, and open custom MBeans and register them with the WebSphere Application Server administrative service.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Do not define new classes as parameters for your MBeans. The classes might not be available in all processes. If you must define a new class, ensure that the class is available on all processes, including the deployment manager, the node agents, and the application servers. If the class is not available for a process, the `ClassNotFoundException` exception occurs for the new class when you attempt to access it.

About this task

Perform the following tasks to create and register a standard, dynamic, or open custom MBean.

1. Create your particular MBean class or classes.
2. Write an MBean descriptor in the XML language for your MBean.
3. Register your MBean by inserting code that uses the WebSphere Application Server run-time `com.ibm.websphere.management.UserMBeanCollaborator` collaborator class into your application code.
4. Package the class files for your MBean interface and implementation, the descriptor XML file, and your application Java archive (JAR) file.

Results

After you successfully complete the steps, you have a standard, dynamic, or open custom MBean that is registered and activated with the WebSphere Application Server administrative service.

Example

The following example shows how to create and register a standard MBean with the WebSphere Application Server administrative service:

SnoopMBean.java:

```
/**
 * Use the SnoopMBean MBean, which has a standard mbean interface.
 */
public interface SnoopMBean {
    public String getIdentification();
    public void snoopy(String parm1);
}
```

SnoopMBeanImpl.java:

```
/**
 * SnoopMBeanImpl - SnoopMBean implementation
 */
public class SnoopMBeanImpl implements SnoopMBean {
    public String getIdentification() {
        System.out.println(">>> getIdentification() called...");
        return "snoopy!";
    }

    public void snoopy(String parm1) {
        System.out.println(">>> snoopy(" + parm1 + ") called...");
    }
}
```

Define the following MBean descriptor for your MBean in an .xml file. The getIdentification method is set to run with the unicast option and the snoopy method is set to use the multicast option. These options are used only for z/OS platform applications. The WebSphere Application Server for z/OS options are not applicable to the distributed platforms, but they do not need to be removed. The options are ignored on the distributed platforms. . Some statements are split on multiple lines for printing purposes.

SnoopMBean.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MBean SYSTEM "MbeanDescriptor.dtd">
<MBean type="SnoopMBean"
    version="5.0"
    platform="dynamicproxy"
    description="Sample SnoopMBean to be initialized inside an EJB.">

    <attribute name="identification" getMethod="getIdentification"
    type="java.lang.String" proxyInvokeType="unicast"/>

    <operation name="snoopy" role="operation" type="void" targetObjectType="objectReference"
    impact="ACTION" proxyInvokeType="multicast">
        <signature>
            <parameter name="parm1" description="test parameter" type="java.lang.String"/>
        </signature>
    </operation>
</MBean>
```

Assume that your MBean is used in an enterprise bean. Register your MBean in the enterprise bean `ejbCreate` method and unregister it in the `ejbRemove` method.

```
//The method MBeanFactory.activateMBean() requires four parameters:
//String type: The type value that you put in this MBean's descriptor. For this example
//the string type is SnoopMBean.
//RuntimeCollaborator co: The UserMBeanCollaborator user MBean collaborator instance
//that you create
//String id: Unique name that you pick
//String descriptor: The MBean descriptor file name
```

```
import com.ibm.websphere.management.UserMBeanCollaborator;
```

```

//Import other classes here.
.
.
.
static private ObjectName snoopyON = null;
static private Object lockObj = "this is a lock";
.
.
.
/**
 * ejbCreate method: Register your Mbean.
 */
public void ejbCreate() throws javax.ejb.CreateException {
    synchronized (lockObj) {
        System.out.println(">>> SnoopMBean activating for --|" + this + "|--");
        if (snoopyON != null) {
            return;
        }
        try {
            System.out.println(">>> SnoopMBean activating...");
            MBeanFactory mbfactory = AdminServiceFactory.getMBeanFactory();
            RuntimeCollaborator snoop = new UserMBeanCollaborator(new SnoopMBeanImpl());
            snoopyON = mbfactory.activateMBean("SnoopMBean", snoop, "snoopMBeanId",
"SnoopMBean.xml");
            System.out.println(">>> SnoopMBean activation COMPLETED! --|" + snoopyON + "|--");
        } catch (Exception e) {
            System.out.println(">>> SnoopMBean activation FAILED:");
            e.printStackTrace();
        }
    }
}
.
.
.
/**
 * ejbRemove method: Unregister your MBean.
 */
public void ejbRemove() {
    synchronized (lockObj) {
        System.out.println(">>> SnoopMBean Deactivating for --|" + this + "|--");
        if (snoopyON == null) {
            return;
        }
        try {
            System.out.println(">>> SnoopMBean Deactivating ==|" + snoopyON + "|== for --|"
+ this + "|--");
            MBeanFactory mbfactory = AdminServiceFactory.getMBeanFactory();
            mbfactory.deactivateMBean(snoopyON);
            System.out.println(">>> SnoopMBean Deactivation COMPLETED!");
        } catch (Exception e) {
            System.out.println(">>> SnoopMBean Deactivation FAILED:");
            e.printStackTrace();
        }
    }
}
}

```

What to do next

Compile the MBean Java files and package the resulting class files with the descriptor .xml file, into the enterprise bean JAR file.

Setting Java 2 security permissions

You must configure Java 2 security permissions to use Java Management Extension and WebSphere Application Server administrative methods.

Before you begin

When you enable Java 2 security, you must grant Java 2 security permissions to application-specific code for Java Management Extensions (JMX) and WebSphere Application Server administrative privileges. With these permissions, your application code can call WebSphere Application Server administrative methods and JMX methods.

About this task

If you are using Java 2 security then you need to verify that your extensions and application server can access the required resources. The following steps show how to configure access for JMX and the application server administrative methods.

- Use the following permission to invoke all the JMX class methods and interface methods:

```
permission javax.management.MBeanPermission "*", "*";
```

Consult the application programming interfaces documentation for specific actions under the `MBeanPermission` class.

- Use the following permission for WebSphere Application Server administrative application programming interfaces (APIs):

```
permission com.ibm.websphere.security.WebSphereRuntimePermission "AdminPermission";
```

Administrative security

Access to the Java Management Extension (JMX) administrative subsystem requires role-based access control when administrative security is enabled.

. A client, which can be a user or an administrative client program, can access an MBean method only if at least one of the required roles is granted to the client. WebSphere Application Server uses the declarative security approach to specify the security policy on the JMX MBean. This approach has the advantage of not requiring MBean developers to add security code. Moreover, WebSphere Application Server provides a default security policy for an MBean so in most case MBean developers do not need to specify a security policy at all. With WebSphere Application Server, you can define explicit security policy for your MBeans if the default security policy does not meet your specific security requirements.

Default MBean security policy

This topic discusses the default managed bean (MBean) security policy. In most cases, MBean developers do not need to specify a security policy.

Three types of MBeans exist for the default MBean security policy:

- A configuration type MBean
- A runtime type MBean
- A deployer type MBean

An optional attribute in the MBean descriptor XML file defines the type of MBean.

The `ConfigRepository` MBean is an example of one of a few configuration types. In the `configRepository.xml` descriptor file, the `configureMBean = "true"` attribute indicates that the MBean is a configuration type.

```
<MBean type="ConfigRepository"
  version="5.0"
  platform="common"
  description="Management interface for the configuration repository."
  configureMBean="true">
```

The `EJBModule` MBean is an example of deployer type MBeans. In the `EJBModule.xml` descriptor file, the `deployerMBean="true"` attribute indicates that the MBean is a deployer type.

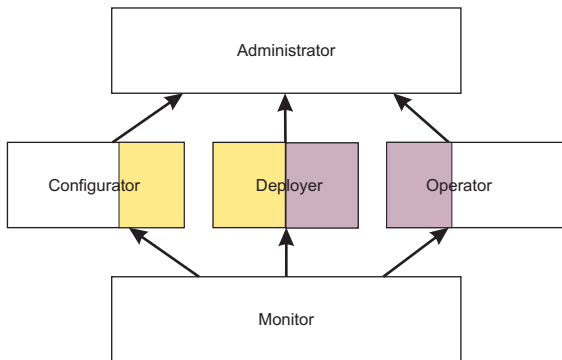
```

<MBean type="EJBModule" j2eeType="EJBModule"
  version="5.0"
  platform="dynamicproxy"
  resourceIdentifierKey="Application"
  resourceType="Application"
  deployerMBean="true"
  description="Management interface for the EJBModule component.">

```

WebSphere Application Server extended role-based access control supports role inheritance. Five administrative roles of administrator, configurator, operator, deployer, and monitor exist. The monitor role is the least privileged administrative role. Users that are granted the monitor role can view the WebSphere Application Server configuration and the runtime status, but cannot make any changes. The other administrative roles each have their own unique set of privileges as well as the same privileges as the monitor role.

The configurator role has permission to modify WebSphere Application Server configuration data. The operator role has permission to change the runtime state, such as the start and stop of administrative resources. A configurator role cannot change the runtime status and conversely an operator role cannot change the WebSphere Application Server configuration. The administrator role includes configurator and operator role, but has more permissions than the union of configurator role and operator role. The administrator role can additionally change the administrative security configuration. A simple picture shows the administrative role inheritance relationship. The deployer role is a combination of the configurator and operator roles for application management. The deployer role has both configurator and operator permission for applications. A diagram shows the administrative role inheritance relationship.



Each MBean method or operation is assigned an impact attribute with a value of either INFO or ACTION. Here are some examples:

- A get method has an impact value of INFO and a write method has an impact value of ACTION.
- In the ConfigRepository MBean, the extract method does not change the configuration data and has an impact value of INFO, while the modify method has an impact value of ACTION.
- In the Java virtual machine (JVM) MBean, which is an operator type of MBean, the ggetCurrentTimeInMillis method has an impact value of ACTION.

A configuration MBean method that has an impact value of INFO requires the monitor role. A configuration MBean method that has an impact value of ACTION requires the configurator role. A deployer MBean method that has an impact value of INFO requires the monitor role. A deployer MBean method that has an impact value of ACTION requires the deployer role. Because all administrative roles are monitor roles, any administrative role can access configuration MBean methods and deployer MBean methods that have an impact value of INFO. The administrator role is a configurator role and has access to the configuration MBean methods that have an impact value of ACTION.

The default security policy for the configuration MBean is summarized in the following table:

Method impact	Monitor role	Operator role	Configurator role	Deployer role	Administrator role
INFO	X	X	X	X	X
ACTION			X		X

The default security policy for the operation MBean is summarized in the following table:

Method impact	Monitor role	Operator role	Configurator role	Deployer role	Administrator role
INFO	X	X	X	X	X
ACTION		X			X

The default security policy for the deployer MBean is summarized in the following table:

Method impact	Monitor role	Operator role	Configurator role	Deployer role	Administrator role
INFO	X	X	X	X	X
ACTION		X		X	X

If an MBean has both the `configureMBean` attribute and the `deployerMBean` attribute set to `true`, the required role for all actions is either configurator or monitor. No such MBean is presently defined in the system.

Defining an explicit MBean security policy

You can explicitly define an MBean security policy for a particular MBean. Use this example to define an MBean security policy.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming, see *MBean Java application programming interface (API) documentation*.

About this task

Perform the following tasks to define an explicit security policy.

1. Assume that you have an MBean defined by the MBean `sample.xml` descriptor file.
2. Specify the explicit security policy for that MBean in the `sampleSecurity.xml` file. The naming convention is that you must append "Security" to the MBean descriptor file name as the name of the MBean security descriptor file.
3. Place the security policy descriptor file at the same directory where the MBean security descriptor file is so that the MBean loader can find it. This directory is the typical location for the security policy descriptor file. If no MBean security descriptor file is present, the default MBean security policy is used.
4. Specify the MBean name of `sample` in the resource element `resource-name` field of the `sampleSecurity.xml` file so that the MBean policy loader can associate the MBean security policy with the MBean. The MBean security descriptor definition is very similar to the security policy that is defined by the Java 2 Platform, Enterprise Edition (J2EE) deployment descriptor.

Results

You now have an explicitly defined MBean security policy that you can run with an MBean.

Example

The following example describes the MBean security descriptor file format for the `sampleSecurity.xml` file.

Line 2 specifies that an MBean security descriptor schema is defined by the `RolePermissionDescriptor.dtd` file, which is a document type definition (DTD) in WebSphere Application Server.

As shown on line 3, each MBean descriptor file contains a single role-permission element. The administrative security role hierarchy is defined in the security-role elements between line 9 and line 37. The administrative security role has an inheritance relationship.

As defined on line 14 through 21, the operator security role implies the monitor security role, which means that a user with the operator role has all the permissions of the monitor role. As defined between line 30 and line 38, an administrator security role implies both the configurator and operator security role. Every MBean security descriptor file typically has the same role relationship definition so that you can cut and paste this section to your MBean security descriptor file.

One or more method-permission elements are defined after the security-role element. Each method-permission element defines the required roles for one or more methods. Specify method parameters to avoid method name collision in case multiple methods have the same name.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE role-permission SYSTEM "RolePermissionDescriptor.dtd" >
3. <role-permission>
4.   <resource>
5.     <resource-name>sample</resource-name>
6.     <class-name>com.ibm.ws.security.descriptor.sample</class-name>
7.     <description>This is a sample for testing role permission descriptor.</description>
8.   </resource>
9.   <security-role>
10.    <role>
11.      <role-name>monitor</role-name>
12.    </role>
13.  </security-role>
14.  <security-role>
15.    <role>
16.      <role-name>operator</role-name>
17.      <imply>
18.        <role-name>monitor</role-name>
19.      </imply>
20.    </role>
21.  </security-role>
22.  <security-role>
23.    <role>
24.      <role-name>configurator</role-name>
25.      <imply>
26.        <role-name>monitor</role-name>
27.      </imply>
28.    </role>
29.  </security-role>
30.  <security-role>
31.    <role>
32.      <role-name>administrator</role-name>
33.      <imply>
34.        <role-name>operator</role-name>
35.        <role-name>configurator</role-name>
36.      </imply>
37.    </role>
38.  </security-role>
39.  <method-permission>
40.    <description>Sample method permission table</description>
41.    <role-name>operator</role-name>
42.    <method>
```

```

43.     <description>Sample operation</description>
44.     <resource-name>sample</resource-name>
45.     <method-name>stop</method-name>
46.   </method>
47. </method-permission>
48. <method-permission>
49.   <description>Sample method permission table</description>
50.   <role-name>operator</role-name>
51.   <method>
52.     <description>Sample operation</description>
53.     <resource-name>sample</resource-name>
54.     <method-name>start</method-name>
55.     <method-params>
56.       <method-param>java.lang.String</method-param>
57.       <method-param>java.lang.String</method-param>
58.     </method-params>
59.   </method>
60. </method-permission>
61. <method-permission>
62.   <description>Sample method permission table</description>
63.   <role-name>operator</role-name>
64.   <method>
65.     <description>Sample operation</description>
66.     <resource-name>sample</resource-name>
67.     <method-name>monitor</method-name>
68.     <method-params>
69.     </method-params>
70.   </method>
71. </method-permission>
72. <method-permission>
73.   <description>Sample method permission table</description>
74.   <role-name>configurator</role-name>
75.   <method>
76.     <description>Sample operation</description>
77.     <resource-name>sample</resource-name>
78.     <method-name>setValue</method-name>
79.     <method-params>
80.       <method-param>java.lang.Boolean</method-param>
81.     </method-params>
82.   </method>
83. </method-permission>
84. <method-permission>
85.   <description>Sample method permission table</description>
86.   <role-name>monitor</role-name>
87.   <method>
88.     <description>Sample operation</description>
89.     <resource-name>sample</resource-name>
90.     <method-name>getValue</method-name>
91.   </method>
92. </method-permission>
93. </role-permission>

```

Specifying fine-grained MBean security in the MBean descriptor

To implement fine-grained administrative security, your code must identify the resource instance that the managed bean (MBean) represents and assign the user the required role for that instance of the resource. This topic discusses what to do to identify the resource and assign the required role. This topic also discusses how to make an MBean method run under a different user identity so that the method can access other resource instances. Lastly, this topic discusses how to check if an MBean method has access to a resource instance by using programmatic interfaces.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming, see MBean Java application programming interface (API) documentation.

About this task

Perform the following task to ensure that an MBean or MBean method is protected. Identify the resource instance that the MBean or MBean method represents and assign required roles to access the MBean. Perform this task during the development of the MBean.

1. Determine the resource instance that the MBean represents and the required roles to invoke the MBean methods.

Every MBean method has a default MBean security policy. When the MBean method uses the default security policy, the resource instance that the MBean represents is assumed to be the server in which the MBean runs. If an MBean or MBean method represents a resource instance other than the server on which it runs, perform the following steps:

- a. Identify the resource instance that the MBean represents.
 - If an MBean, such as the Server MBean, accesses and modifies the server in which the MBean runs, do not specify a security policy to verify that the user invoking the MBean is granted access to the server because the default security policy is in force. In most cases, you use an MBean to access and modify the server.
 - If an MBean that runs inside a server can access and modify resources that do not directly belong to the server, check if the user invoking the MBean is granted access to the instance of the resource before allowing the MBean method to run.

In most cases, identify the resource instance by identifying the key-value pair in the object name of the MBean that represents the resource instance. The `resourceIdentifierKey` attribute defines the key.

For example, you can use the `EJBModule` MBean to access an Enterprise JavaBeans (EJB) module within an application that runs inside the server. In this case, the object name of the `EJBModule` MBean contains a key-value pair. The key is `Application`. The value represents the resource instance that the `EJBModule` MBean tries to access. The user that invokes this MBean method is verified to make sure that access is granted to this instance of the application before allowing the MBean method to run.

The following example shows how to describe the fine-grained administrative security for the `EJBModule` type of MBean in the MBean descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MBean SYSTEM "MbeanDescriptor.dtd">
<MBean type="EJBModule" j2eeType="EJBModule"
  version="5.0"
  platform="dynamicproxy"
  resourceIdentifierKey="Application"
  resourceType="Application"
  deployerMBean="true"
  description="Management interface for the EJBModule component.">
```

- If you can determine the resource that the MBean accesses before the MBean is invoked, but you cannot use the MBean object name to determine the resource instance that the MBean accesses, use parameters that are passed to the MBean instead.

Identify the MBean method parameter name with a parameter value that represents the resource instance. Mark the corresponding parameter metadata in the MBean descriptor as the resource identifier. To mark a parameter as the resource identifier, add the `resourceType` attribute. The attribute specifies the type resource that the parameter value contains. When the `resourceType` attribute is present for any MBean method parameter, the parameter value determines the resource instance that the MBean method represents.

For example, one instance of the `ApplicationManager` MBean runs in each server. The same MBean can be used to start and stop all the applications in the server. The start and stop methods of this MBean each take the application name as a parameter. They use the parameter to determine the instance of the application that this MBean method tries to access.

The following example shows how to describe the fine-grained administrative security for this type of MBean in the MBean descriptor:


```

<operation
  description="Start Application"
  impact="ACTION" name="startApplication" role="operation"
  targetObjectType="objectReference" type="void" proxyInvokeType="spray">
  <signature>
    <parameter description="Application Name" resourceType="Application"
      name="applicationName" type="java.lang.String"/>
  </signature>
</operation>

```

- If the resource that an MBean accesses cannot be determined until the MBean is invoked, check if the user invoking the MBean is granted access to the instance of the resource by using application programming interfaces (APIs).

Mark the MBean or MBean method as excluded from access checking in the MBean descriptor by using the `excludeAccessCheck` attribute. When an MBean is marked as excluded from access checking, all its methods are also excluded from access checking.

For example, the `ConfigService` MBean that runs in the deployment manager is used to configure all the resources within a cell. Exclude this MBean from access checking before invoking the MBean methods. Check that the `ConfigService` MBean is granted access to the configuration resource when the MBean attempts to access the resource.

The following example shows how to describe the fine-grained administrative security for the `ConfigServices` type of MBean in the MBean descriptor:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MBean SYSTEM "MbeanDescriptor.dtd">
<MBean
  version="5.0"
  platform="proxy"
  collaboratorClass="com.ibm.ws390.management.proxy.ConfigServiceManager"
  description="Config Service component provides service of
configuration related tasks on top of configuration repository service."
  type="ConfigService"
  excludeAccessCheck="true"
  configureMBean="true">

```

Some statements are split on multiple lines for printing purposes.

The following example shows how to invoke the MBean method logic to perform authorization checking programmatically:

```

// Get administration authorizer.
AdminAuthorizer aa = AdminAuthorizerFactory.getAdminAuthorizer();
// Set the role that is required for this operation.
String role = com.ibm.ws.security.util.Constants.CONFIG_ROLE;
// Set the resource name.
// cells/cellName is optional.
String resource = "/nodes/" + nodeName + "/servers/" + serverName;
// Check access
if ( aa != null && !aa.checkAccess(resource, role) )
  // Disallow access.
else
  // Allow access.

```

- b. Assign required roles for the MBean and MBean methods.

The required roles are automatically assigned, based on the type of MBean and the impact of the MBean method, as described in the topic on the default MBean security policy.

2. Specify delegation mode.

In some cases, after performing the initial access check, the MBean method might need to run under a different user identity so that it can access other resource instances. For example the `syncNode` operation in the `CellSync` MBean grants the user the operator role to the instance of the node being synchronized. The `syncNode` operation tries to access resources under the cell scope. The user might not have access to open files under the cell directory. The MBean must run as `System` after the initial access check so that the operation completes without any access denied problems.

Set the runAs attribute to System to specify delegation mode for an MBean or MBean method. When you set the runAs attribute for an MBean, the value applies to all MBean methods for that MBean. The following example shows how to describe fine-grained administrative security for the CellSync type of MBean in the MBean descriptor.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MBean SYSTEM "MbeanDescriptor.dtd">
<MBean type="CellSync"
  version="5.0.1"
  platform="common"
  runAs="System"
  description="Management interface for the configuration synchronization logic
  performed at the central deployment manager for the cell.">
  <operation
    description="Initiate a synchronization request for a given node" impact="ACTION"
    name="syncNode" role="operation" targetObjectType="objectReference" type="ja
va.lang.Boolean">
  <signature>
  <parameter resourceType="Node"
    description="The name of the node"
    name="nodeName" type="java.lang.String"/>
  </signature>
</operation>
```

Results

You have determined the type of resource a given MBean method is accessing and performed the necessary access check so that WebSphere Application Server can allow access to the resource.

Administrative programs for multiple Java Platform, Enterprise Edition application servers

You can develop an administrative client to manage multiple vendor application servers through existing MBean support in the WebSphere Application Server.

Existence of MBeans for stopped components

Note: The WebSphere Application Server completely implements the Java Platform, Enterprise Edition (Java EE) Management specification. However, some differences in details between the Java EE specification and the WebSphere Application Server implementation are important for you to understand when you access WebSphere Application Server components. These differences are important to you when you access application MBeans because you can use either the WebSphere Application Server programming model or the Java EE programming model.

In the WebSphere Application Server programming model, if an MBean exists, you can assume that it is running. If an MBean does not exist, you can assume that it is stopped. Transient states between the started state and the stopped state are the same as the stopped state, which means that no MBean exists.

In the Java EE programming model, the MBean always exists regardless of the state of the component.

You can determine the state of a component by querying the state attribute. However, the state attribute only exists for MBeans that are state manageable, meaning that they implement the StateManageable interface. State manageable MBeans have start(), startRecursive(), and stop() operations whether these MBeans are Java EE MBeans or WebSphere Application Server MBeans. Additionally, the WebSphere Application Server defines the stateful interface. The stateful interface means that the component has a state and emits the Java EE.state.notifications method, but that the component cannot directly manage the state. For example, a Web module cannot stop itself. However, the application that contains the Web module can stop it.

Not all MBeans that have a state are state-manageable. Servlets, Java EE modules and enterprise beans, for example, are all stateful, but are not state manageable. The Java EE server is not state-manageable because no start() operation is available on a server.

The J2EEApplication MBean is an example of a state manageable MBean. When the WebSphere Application Server starts, each application activates a J2EEApplication MBean for itself. A J2EEApplication MBean has a Java EE type of J2EEApplication (for example, `ObjectName **:*,j2eeType=J2EEApplication`). If the application starts, it also activates an Application MBean with a type of Application (for example, `**:*,type=Application`). When the application changes state, the Application MBean is activated or deactivated. However, the J2EEApplication MBean is always activated. You can retrieve the application state changes by getting the state attribute.

The modules attribute on the J2EEApplication component returns an array of object names, one for every module in the application. The Application Server activates an MBean for each of these modules only after the Application Server starts the application. The managed enterprise bean `isRegistered(ObjectName)` method returns `false` if the application, and therefore the module, is not running.

All of the attributes that are defined in the Java EE management specification return valid values when the managed object stops. Other attributes and operations, for example those that are specifically defined for the Application Server, use the `com.ibm.websphere.management.exception.ObjectNotRunningException` exception if they are accessed when the object is stopped.

If you install the application while the server runs, the application installs the J2EEApplication MBean when the installation completes. Conversely, when the application uninstalls the J2EEApplication MBean, the application deactivates the MBean.

Mapping type properties

You can determine which MBeans have a `j2eeType` property and a WebSphere Application Server type property in their `ObjectName` property sets by going to the additional application programming interfaces documentation. Select the MBean interfaces subtopic, and then specific MBeans in the list. Look for `type=` and `j2eeType=`. You can use the type property to query for any MBeans. MBeans derived from the Java EE specification have an additional `j2eeType` property as part of their `ObjectName` property sets. You can also use the `j2eeType` property to query for MBeans.

Optional WebSphere Application Server interfaces

The `EventProvider`, `StateManageable`, and `StatisticsProvider` interfaces are optional interfaces that the Java EE Management specification defines. Which of the interfaces the product implements varies from MBean to MBean. Go to the additional application programming interfaces documentation to see which interfaces the product implements for a particular MBean. Select the MBean interfaces subtopic, and then a specific MBean in the list. Find All Parent MBeans. The interfaces that are implemented for the MBean follow All Parent MBeans. For example, the `J2EEDomain` MBean does not implement any of the interfaces, while the `JVM` MBean implements the `StatisticsProvider` interface.

Deploying and managing a custom Java administrative client program with multiple Java Platform, Enterprise Edition application servers

This section describes how to connect to a Java Platform, Enterprise Edition (Java EE) server, and how to manage multiple vendor servers.

Before you begin

The WebSphere Application Server completely implements the Java EE Management specification, also known as JSR-77 (Java Specification Requests 77). However, some differences in details between the Java EE specification and the WebSphere Application Server implementation are important for you to

understand when you develop a Java administrative client program to manage multiple vendor servers. For information, see the Java Platform, Enterprise Edition (Java EE) Management Specification and the MBean Java application programming interface (API) documentation.

About this task

When your administrative client program accesses WebSphere Application Servers exclusively, you can use the Java APIs and WebSphere Application Server-defined MBeans to manage them. If your program needs to access both WebSphere Application Servers and other Java EE servers, use the API defined in the Java EE Management specification.

1. Connect to a Java EE server.

Connect to a server by looking up the Management enterprise bean from the Java Naming and Directory Interface (JNDI). The Management enterprise bean supplies a remote interface to the MBean server that runs in the application server. The Management enterprise bean works almost exactly like the WebSphere Application Server administrative client, except that it does not provide WebSphere Application Server specific functionality. The following example shows how to look up the Management enterprise bean.

```
import javax.management.j2ee.ManagementHome;
import javax.management.j2ee.Management;

Properties props = new Properties();

props.setProperty(Context.PROVIDER_URL, "iiop://myhost:2809");
Context ic = new InitialContext(props);
Object obj = ic.lookup("ejb/mgmt/MEJB");
ManagementHome mejbHome = (ManagementHome)
    PortableRemoteObject.narrow(obj, ManagementHome.class);
Management mejb = mejbHome.create();
```

The example gets an initial context to an application server by passing the host and port of the Remote Method Invocation (RMI) connector. You must explicitly code the RMI port, in this case 2809. The lookup method looks up the `ejb/mgmt/MEJB` path, which is the location of the Management enterprise bean home. The example then creates the `mejb` stateless session bean, which you use in the next step.

2. Manage multiple vendor application servers.

After you create the `mejb` stateless session bean, you can use it to manage your application servers. Components from the application servers appear as MBeans, which the specification defines. These MBeans all have the `j2eeType` property. This property is one of a set of types that the specification defines. All of these types have a set of exposed attributes.

Use the following example to guide you in managing multiple vendor application servers. The example uses the Java virtual machine (JVM) MBean to determine what the current heap size is for the application server.

```
ObjectName jvmQuery = new ObjectName("*:j2eeType=JVM,*");
Set s = mejb.queryNames(jvmQuery, null);
ObjectName jvmMBean = (ObjectName) s.iterator().next();
boolean hasStats = ((Boolean) mejb.getAttribute(jvmMBean,
    "statisticsProvider")).booleanValue();
if (hasStats) {
    JVMStats stats = (JVMStats) mejb.getAttribute(jvmMBean,
        "stats");
    String[] statisticNames = stats.getStatisticNames();
    if (Arrays.asList(statisticNames).contains("heapSize")) {
        System.out.println("Heap size: " + stats.getHeapSize());
    }
}
```

The `queryNames()` method first queries the JVM MBean. The `getAttribute` method gets the `statisticsProvider` attribute and determine if this MBean provides statistics. If the MBean does, the example accesses the `stats` attribute, and then invokes the `getHeapSize()` method to get the heap size.

Results

The strength of this example is that the example can run on any vendor application server. It demonstrates that an MBean can optionally implement defined interfaces, in this case the `StatisticsProvider` interface. If an MBean implements the `StatisticsProvider` interface, you can see if an application server supports a particular statistic, in this case the heap size. The specification defines the heap size, although this value is optional. If the application server supports the heap size, you can display the heap size for the JVM.

Java Management Extensions V1.0 to Java Management Extensions V1.2 migration

You might need to migrate custom MBeans that are supplied by products other than the Application Server from Version 5 to Version 6.0 and later for full compatibility.

Each Java virtual machine (JVM) in WebSphere Application Server includes an embedded implementation of Java Management Extensions (JMX). In Application Server, Version 5, the JVMs contain an implementation of the JMX 1.0 specification. In Application Server, Version 6.0 and later, the JVMs contain an implementation of the JMX 1.2 specification. The JMX 1.0 implementation used in Version 5 is the `TMX4J` package that IBM Tivoli products supply. The JMX 1.2 specification used in Version 6.0 and later is the open source `mx4j` package. The JMX implementation change across the releases does not affect the behavior of the JMX MBeans in the Application Server. No Application Server administrative application programming interfaces (APIs) are altered due to the change from the JMX V1.0 specification to the JMX V1.2 specification.

The JMX V1.2 specification is compatible with the earlier JMX V1.0 specification. However, you might need to migrate custom MBeans that are supplied by products other than the application server from Version 5 to Version 6.0 and later. The primary concern for these custom MBeans is related to the values that are used in key properties of the `JMX ObjectName` class for the MBean. The open source `mx4j` implementation more stringently enforces property validation according to the JMX 1.2 specification. Test the custom MBeans that you deployed in Version 5 in Version 6.0 and later, to ensure compatibility. Full details of the JMX V1.2 specification changes from the JMX V1.0 specification are available in the JMX 1.2 specification.

Java Management Extensions interoperability

Starting with Version 6.0, WebSphere Application Server implements Java Management Extensions (JMX) Version 1.2, while WebSphere Application Server Version 5 implements JMX Version 1.0.

Differences between Version 5 and Version 6.0 and later

Due to the evolution of the JMX specification, the serialization format for JMX objects, such as the `javax.management.ObjectName` object, differs between the Version 5 implementation and the Version 6.0 and later implementation. The Version 6.0 and later JMX run time is enhanced to be aware of the version of the client with which it is communicating. The Version 6.0 and later run time makes appropriate transformations on these incompatible serialized formats to support communication between the different version run times.

A Version 5 `wsadmin` script or a Version 5 administrative client can call a Version 6.0 and later server. A Version 6.0 and later `wsadmin` script or a Version 6.0 and later administrative client can call a Version 5 server.

When a Version 5 `wsadmin` script or a Version 5 administrative client calls a Version 6.0 and later MBean, the instances of classes that are new in Version 6.0 and later cannot be passed back to Version 5 because these classes are not present in the Version 5 environment. The problem occurs infrequently. However, it usually occurs when an exception embeds a nested exception that is new starting with Version 6.0. The symptom is usually a serialization exception or a `NoClassDefFoundException` exception.

Due to changes in the JMX implementation from Version 5 to Version 6.0, different exceptions are created when a method on an MBean is invoked for Version 5 than when a method on an MBean is invoked for Version 6.0 and later. For example, when a method gets or sets an unknown attribute for Version 5, the `MBeanRuntimeException` exception is created. When a method gets or sets an unknown attribute for Version 6.0 and later, the `MBeanException` exception that wraps a `ServiceNotFoundException` exception is created.

An instance of a user-defined class that implements the `Serializable` interface that is passed as a parameter or return value during MBean invocation, or sent as part of a notification, cannot contain a non-transient instance variable that is in the `javax.management.package` package. If the instance does, it cannot be properly deserialized when passed between Version 5 and Version 6.0 or later run times.

Due to changes in the supported format for the `ObjectName` class from Version 5 to Version 6.0 and later, the configuration ID in starting with Version 6.0 and later contains a vertical bar (`|`), whereas in Version 5, the ID contains a colon (`:`). This change is reflected in the output for `wsadmin` clients. For example, for a Version 5 client, the output is:

```
wsadmin> $AdminConfig list Cell
      DefaultCellNetwork(cells/DefaultCellNetwork:cell.xml#Cell_1)
```

whereas for a Version 6.0 and later client, the output is:

```
wsadmin> $AdminConfig list Cell
      DefaultCellNetwork(cells/DefaultCellNetwork|cell.xml#Cell_1)
```

The change to the configuration ID generally is not a problem because configuration IDs are generated dynamically. When a Version 5 client passes a configuration ID that contains a colon, the JMX run time, for upward compatibility, automatically transforms the configuration ID that contains a colon into a configuration ID that contains a vertical bar. Similarly, a reverse transformation is performed for backward compatibility.

Do not save the configuration ID and then try to use it later. Only query the ID and use it.

Differences between Version 6.0.x, and Version 6.1 and later

A serialization format mismatch exists between the JMX implementation in Version 6.1 and later and Version 6.0.x releases. When a Version 6.0.x `wsadmin` script or a Version 6.0.x administrative client tries to retrieve the `ModelMBeanInfo` interface of a Version 6.1 and later MBean, the expected field names are not found in the deserialized object because of the case difference between the versions. For example, the following `wsadmin` function does not work when a Version 6.0.x `wsadmin` script connects to a Version 6.1 and later server:

```
$Help attributes MBeanObjectName
$Help operations MBeanObjectName
$Help all MBeanObjectName
```

where `MBeanObjectName` is a string representation of a Version 6.1 and later MBean Object.

To avoid this problem, set the `jmx.serial.form` Java virtual machine (JVM) custom property on the JVM custom properties page in the administrative console. Create the custom property by specifying the name value-pair on the Version 6.1 and later Application Server that you are connecting to from a Version 6.0.x client. The field names are forced to lower case to be compatible with what the Version 6.0.x client expects. The lower case field names contradict the JMX specification and compromise interoperability with future versions. Therefore, the recommendation is that you set this property only when it is absolutely needed in a mixed version environment.

Property name	<code>jmx.serial.form</code>
Data type	string
Value	1.2.0 or 1.2.1

To access the JVM custom properties page, click:

Servers > Server Types > WebSphere application servers > *server1*. . Then, under Server Infrastructure, click **Java and process management > Process Definition > Java virtual machine > Custom properties**

Using command line tools

WebSphere Application Server provides many tools that you can call from a command line.

About this task

There are several command line tools that you can use to start, stop, and monitor application server processes and nodes. These tools only work on local servers and nodes. They cannot operate on a remote server or node. To administer a remote server, you can use the wsadmin scripting program connected to the deployment manager for the cell in which the target server or node is configured.

The following information is common for each command:

- Unless otherwise specified, all of the application server commands are located in the *app_server_root/bin* directory.
- Parameter values that specify a server name, a node name or a cell name are case sensitive. For example, if you want to start the application server MyServer for the profile test, invoke `startServer MyServer -profileName test`. If you specify `myserver` for the server name, the `startServer` script fails.

Use the following general steps to run a command:

1. From the i5/OS command line, start a Qshell session by issuing the **STRQSH CL** command.
2. Determine whether to run the script from the profile or application server root directory.

Most command line tools function relative to a particular profile. To determine if a command requires the `-profileName` parameter, refer to the documentation for that specific command. If you run a command from the *app_server_root/bin* directory and do not specify the `-profileName` parameter, the default profile for the product sets profile-specific variables. To specify a different profile, use one of the following options:

- Navigate to the *app_server_root/bin* directory and run the following command, specifying the profile of interest as the value for the `profileName` parameter:

```
startServer server1 -profileName AppServerProfile
```

- When a profile is created, the application server creates a proxy script in the *profile_root/bin* directory for each script in the *app_server_root/bin* directory that is applicable to the type of profile created. When a proxy script is invoked, the profile-specific variables for the script are set based on the profile from which the script is invoked. To run the command for a specific profile, navigate to the *profile_root/bin* directory for the profile of interest to run the command.

3. Run the command of interest.

Results

The command runs the requested function and displays the results on the screen.

Refer to the command log file for additional information. When you use the `-trace` option for the command, the additional trace data is captured in the command log file. The directory location for the log files is under the default system log root directory, except for commands related to a specific server instance, in which case the log directory for that server is used. You can override the default location for the command log file using the `-logfile` option for the command.

What to do next

For more information about using profiles, including how to obtain a list of profiles, see the `manageprofiles` command article.

To use the command line tools, perform the following steps:

Configuring Qshell to run WebSphere Application Server scripts

You can configure Qshell to run WebSphere Application Server for i5/OS scripts.

About this task

In a default installation, unless otherwise specified, all WebSphere Application Server scripts are in the `app_server_root/bin` directory.

The scripts must be run from Qshell. To ensure that you use the correct version and directory, run QShell commands using one of the following methods:

- Invoke the fully qualified path name of the script:

```
app_server_root/bin/script_name parameters
```

where `script_name` is the name of the script and `parameters` represents the parameters that are passed to the script.

- Invoke the script from the i5/OS command line or from an i5/OS CL program. To use this method, run the STRQSH command and specify the fully qualified path name of the script:

```
STRQSH CMD('app_server_root/bin/script_name parameters')
```

where `script_name` is the name of the script and `parameters` represents the parameters that are passed to the script.

- Change directories to the `app_server_root/bin` directory and run the following script:

```
cd app_server_root/bin
script_name parameters
```

where `script_name` is the name of the script, and `parameters` represents the parameters that are passed to the script.

- You can update the PATH environment variable to automatically locate the script when you run it. After you update the PATH variable, you can run these scripts from any directory. To update the PATH environment variable, perform the following steps:

1. Edit the `.profile` file in the `/home/user_profile_name` directory, where `user_profile_name` is the name of your iSeries user profile.

If this file does not exist, create it in this directory. You can use the EDTF command from an i5/OS command line or use any editor from a workstation. Also note that `.profile` is the full name of the file. When you start Qshell, it searches for the `.profile` file, and runs the commands listed in it. You can use the `.profile` file to set persistent environment variables for your Qshell session.

2. Add the following line to the `.profile` file:

```
export PATH=app_server_root/bin:$PATH
```

3. Save the file.

What to do next

The updates to `.profile` do not take effect until you restart Qshell (if you had a session open) or source `.profile` (command: `./home/user_profile_name/.profile`)

To change the path for a single Qshell session instead of changing the default path for all Qshell sessions, run the **export** commands shown in the last step during the Qshell session, but do not edit the `.profile`.

Qshell environment variables

WebSphere Application Server provides these Qshell environment variables that affect the WebSphere Application Server scripts.

To set these variables, run the following command:

```
export variableName=value
```

To unset these variables, run the following command:

```
unset variableName
```

- **WAS_ADDL_JVM_ARGS**

If you set this variable, its value is appended to the Java virtual machine (JVM) arguments for all of the scripts that run a JVM. All of the commands except `deleteProdLod`, `detectprocess`, `enbprfwasm`, `grtwasm`, and `rvkwasm` run a JVM. For example, this command:

```
export WAS_ADDL_JVM_ARGS="-Dtrace=com.ibm.*=all=enabled -Xms256m"
```

enables tracing for the JVM that a script starts, and sets the minimum heap size for the JVM to 256 megabytes.

- **WAS_USER_SCRIPT**

This variable specifies the path to a script that runs before a WebSphere Application Server Qshell script runs. For example, if you want to append a `/home/QSYS/classes.jar` to the classpath for every script, you might create a script called `/home/myDir/classpath.script`. The script file contains this command:

```
WAS_CLASSPATH=${WAS_CLASSPATH}:/home/QSYS/classes.jar
```

After you create the script file, run this command to set the environment variable:

```
export WAS_USER_SCRIPT=/home/myDir/classpath.script
```

Example: Security and the command line tools

If you want to enable WebSphere Application Server security, you need to provide the command line tools with authentication information.

Without authentication information, the command line tools receive an `AccessDenied` exception when you attempt to use them with security enabled. There are multiple ways to provide authentication data:

- Most command line tools support a `-username` and `-password` option for providing basic authentication data. Specify the user ID and password for an administrative user. For example, you can use a member of the administrative console users with operator or administrator privileges, or the administrative user ID configured in the user registry. The following example demonstrates the **stopNode** command, which specifies command line parameters:

```
stopNode -username adminuser -password adminpw
```
- You can place the authentication data in a properties file that the command line tools read. The default file for this data is the `sas.client.props` file in the `properties` directory for the current profile.

manageprofiles command

Use the **manageprofiles** command to create, delete, augment, backs up, and restore profiles, which define runtime environments. Using profiles instead of multiple product installations saves disk space and simplifies updating the product because a single set of core product files is maintained.

The command file is located in the `app_server_root/bin` directory. The command file is a script named `manageprofiles`.

For more information about where to run this command, see Using command line tools.

Note: If you use this command with the managed profile template, application servers are not created. However, ports are still used if you are federating a node.

Syntax

The **manageprofiles** command is used to perform the following tasks:

- create a profile (-create)
- delete a profile (-delete)
- augment a profile (-augment)
- unaugment a profile (-unaugment)
- unaugment all profiles that have been augmented with a specific augmentation template (-unaugmentAll)
- delete all profiles (-deleteAll)
- list all profiles (-listProfiles)
- list augments for a profile (-listAugments)
- get a profile name (-getName)
- get a profile path (-getPath)
- validate a profile registry (-validateRegistry)
- validate and update a profile registry (-validateAndUpdateRegistry)
- get the default profile name (-getDefaultName)
- set the default profile name (-setDefaultName)
- back up a profile (-backupProfile)
- restore a profile (-restoreProfile)
- perform **manageprofiles** command tasks that are contained in a response file (-response)

For detailed help including the required parameters for each of the tasks accomplished with the **manageprofiles** command, use the `-help` parameter. The following example uses the help parameter with the **manageprofiles** `-augment` command on Windows operating systems:

```
app_server_root\bin\manageprofiles.bat -augment -help
```

The output from the help command will specify which parameters are required and which are optional.

Depending on the operation that you want to perform with the **manageprofiles** command, you need to provide one or more of the following parameters. The command line profile management tool validates that the required parameters are provided and the values entered for those parameters are valid. Be sure to type the name of the parameters with the correct upper and lower case as the command line profile management tool does not validate the case of the parameter name. Incorrect results can occur when the parameter case is not typed correctly.

- `-profileName` *profile_name*
- `-profilePath` *profile_root*
- `-templatePath` *template_path*
- `-nodeName` *node_name*
- `-cellName` *cell_name*
- `-hostName` *host_name*
- `-serverName` *server_name*
- `-adminUserName` *adminUser_ID*
- `-adminPassword` *adminPassword*

- -backupFile *backupFile_name*
- -debug
- -enableAdminSecurity true | false
- -enableAdminSecurity true | false
- -federateLater true | false
- -importPersonalCertKS *keystore_path*
- -importPersonalCertKSType *keystore_type*
- -importPersonalCertKSPassword *keystore_password*
- -importPersonalCertKSAlias *keystore_alias*
- -importSigningCertKS *keystore_path*
- -importSigningCertKSType *keystore_type*
- -importSigningCertKSPassword *keystore_password*
- -importSigningCertKSAlias *keystore_alias*
- -isDefault
- -isDeveloperServer
- -keyStorePassword *keystore_password*
- -listAugments
- -omitAction *feature1 feature2... featureN*
- -personalCertDN *distinguished_name*
- -personalCertValidityPeriod *validity_period*
- -response *response_file*
- -samplesPassword *samplesPassword*
- -serverType ADMIN_AGENT
- -signingCertDN *distinguished_name*
- -signingCertValidityPeriod *validity_period*
- -startingPort *starting_port* | -portsFile *file_path* | -defaultPorts
- -unaugmentAll
- -unaugmentDependents true | false
- -validatePorts
- -webServerCheck true | false
- -webServerHostname *webserver_host_name*
- -webServerInstallPath *webserver_installpath_name*
- -webServerName *webserver_name*
- -webServerOS *webserver_operating_system*
- -webServerPluginPath *webserver_plugin_path*
- -webServerPort *webserver_port*
- -webServerType *webserver_type*

The following example uses the **manageprofiles** -create command on operating systems such as AIX® or Linux®:

```
app_server_root/bin/manageprofiles.sh -create
    -profileName profile_name
    -profilePath profile_root
    -templatePath template_path
```

Parameters

The following options are available for the **manageprofiles** command:

-adminUserName *adminUser_ID*

Specify the user ID that is used for administrative security.

-adminPassword *adminPassword*

Specify the password for the administrative security user ID specified with the **-adminUserName** parameter.

-augment

Use the **augment** parameter to make changes to an existing profile with an augmentation template. The **augment** parameter causes the **manageprofiles** command to update or augment the profile identified in the **-profileName** parameter using the template in the **-templatePath** parameter. The augmentation templates that you can use are determined by which IBM products and versions are installed in your environment.

Note: The templates that are included with WebSphere Application Server can only be used to create profiles and not to augment existing profiles because only create templates are shipped with the product.

Also, do not manually modify the files that are located in the *install_dir/profileTemplates* directory. For example, if you are changing the ports during profile creation, use the Profile Management Tool or the **-startingPort** or **-portsFile** arguments on the **manageprofiles** command instead of modifying the file in the profile template directory.

You can specify a relative path for the **-templatePath** parameter if the profile templates are relative to the *app_server_root/profileTemplates* directory. Otherwise, specify the fully qualified template path. For example:

```
manageprofiles -augment -profileName profile_name -templatePath template_path
```

See also the **-unaugment** parameter.

-backupProfile

Performs a file system backup of a profile folder and the profile metadata from the profile registry file. Any servers using the profile that you want to back up must first be stopped prior to invoking the **manageprofiles** command with the **-backupProfile** option. The **-backupProfile** parameter must be used with the **-backupFile** and **-profileName** parameters, for example:

```
manageprofiles(.bat)(.sh) -backupProfile -profileName profile_name -backupFile backupFile_name
```

When you back up a profile using the **-backupProfile** option, you must first stop the server and the running processes for the profile that you want to back up.

-backupFile *backupFile_name*

Backs up the profile registry file to the specified file. You must provide a fully qualified file path for the *backupFile_name*.

-cellName *cell_name*

Specifies the cell name of the profile. Use a unique cell name for each profile.

This is an optional parameter. If you omit the parameter, a default cell name is assigned.

The default cell names are as follows:

- dmgr template: *profilenameNetwork*
- default template: *shorthostname_profilename*
- managed template: *shorthostname_profilename*
- cell template: Same as the previous dmgr example for the two profiles that are created.

The value for this parameter must not contain spaces or any invalid characters that are not valid such as the following: *, ?, ", <, >, ,, /, \, |, and so on.

-create

Creates the profile.

Specify `manageprofiles -create -templatePath fully_qualified_file_path_to_template -help` for specific information about creating a profile. Available templates include:

- **management** - Management. Use in conjunction with the `-serverType` parameter to indicate the type of management profile.
- **default** - Application server

-debug

Turns on the debug function of the Ant utility, which the **manageprofiles** command uses.

-personalCertValidityPeriod *validity_period*

An optional parameter that specifies the amount of time in years that the default personal certificate is valid. If you do not specify this parameter with the `-personalCertDN` parameter, the default personal certificate is valid for one year.

-defaultPorts

Assigns the default or base port values to the profile.

Do not use this parameter when using the `-startingPort` or `-portsFile` parameter.

During profile creation, the **manageprofiles** command uses an automatically generated set of recommended ports if you do not specify the `-startingPort` parameter, the `-defaultPorts` parameter or the `-portsFile` parameter. The recommended port values can be different than the default port values based on the availability of the default ports.

Note: Do not use this parameter if you are using the managed profile template.

-delete

Deletes the profile.

The profile directory is deleted when you delete the profile so that you can recreate the profile without having to manually delete the directory.

If you delete a profile that has augmenting templates registered to it in the profile registry, then unaugment actions are performed automatically.

-deleteAll

Deletes all registered profiles.

The directory for the profile is deleted when you delete the profile so that when you recreate the profile, you do not have outdated information to manage.

If you delete a profile that has augmenting templates registered to it in the profile registry, then unaugment actions are performed automatically.

-enableAdminSecurity *true* | *false*

Enables administrative security. Valid values include `true` or `false`. The default value is `false`.

When `enableAdminSecurity` is set to `true`, you must also specify the parameters `-adminUserName` and `-adminPassword` along with the values for these parameters. If samples were installed during the application server install, you must also specify the `-samplesPassword` parameter when creating a profile for which administrative security is enabled. If the `-samplesPassword` parameter is not specified when administrative security is enabled, the profile is created successfully, but when you attempt to run the samples, exceptions and failures will be put in the server system out log.

You cannot use the `-enableAdminSecurity` parameter to enable administrative security for a custom profile. For security to be enabled for a custom profile, the custom profile must be federated into a deployment manager. Administrative security enabled for the deployment manager is required to enable security for the federated custom profile.

-federateLater true | false

Indicates if the managed profile will be federated during profile creation or if you will federate it later using the **addNode** command. If the `dmgrHost`, `dmgrPort`, `dmgrAdminUserName` and `dmgrAdminPassword` parameters do not have values, the default value for this parameter is `true`. Valid values include `true` or `false`.

-getDefaultName

Returns the name of the default profile.

-getName

Gets the name for a profile registered at a given `-profilePath` parameter.

-getPath

Gets the file system location for a profile of a given name. Requires the `-profileName` parameter.

-help

Displays command syntax.

-hostName *host_name*

Specifies the host name where you are creating the profile. This should match the host name that you specified during installation of the initial product. The default value for this parameter is the long form of the domain name system. The value for this parameter must be a valid IPv6 host name and must not contain spaces or any characters that are not valid such as the following: `*`, `?`, `"`, `<`, `>`, `,`, `/`, `\`, `|`, and so on.

-ignoreStack

An optional parameter that is used with the `-templatePath` parameter to unaugment a particular profile that has been augmented. See the `-unaugment` parameter.

-importPersonalCertKS *keystore_path*

Specifies the path to the keystore file that you use to import a personal certificate when you create the profile. The personal certificate is the default personal certificate of the server.

Note: When you import a personal certificate as the default personal certificate, import the root certificate that signed the personal certificate. Otherwise, the `manageprofiles` adds the public key of the personal certificate to the `trust.p12` file and creates a root signing certificate.

The `-importPersonalCertKS` parameter is mutually exclusive with the `-personalCertDN` parameter. If you do not specifically create or import a personal certificate, one is created by default.

When you specify any of the parameters that begin with `-importPersonal`, you must specify them all.

-importPersonalCertKSType *keystore_type*

Specifies the type of the keystore file that you specify on the `-importPersonalCertKS` parameter. Values might be `JCEKS`, `CMSKS`, `PKCS12`, `PKCS11`, and `JKS`. However, this list can change based on the provider in the `java.security` file.

When you specify any of the parameters that begin with `-importPersonal`, you must specify them all.

-importPersonalCertKSPassword *keystore_password*

Specifies the password of the keystore file that you specify on the `-importPersonalCertKS` parameter.

When you specify any of the parameters that begin with `-importPersonal`, you must specify them all.

-importPersonalCertKSAlias *keystore_alias*

Specifies the alias of the certificate that is in the keystore file that you specify on the `-importPersonalCertKS` parameter. The certificate is added to the server default keystore file and is used as the server default personal certificate.

When you specify any of the parameters that begin with `-importPersonal`, you must specify them all.

-importSigningCertKS *keystore_path*

Specifies the path to the keystore file that you use to import a root certificate when you create the profile. The root certificate is the certificate that you use as the server default root certificate. The

`-importSigningCertKS` parameter is mutually exclusive with the `-signingCertDN` parameter. If you do not specifically create or import a root signing certificate, one is created by default.

When you specify any of the parameters that begin with `-importSigning`, you must specify them all.

-importSigningCertKSType *keystore_path*

Specifies the type of the keystore file that you specify on the `-importSigningCertKS` parameter. Valid values might be JCEKS, CMSKS, PKCS12, PKCS11, and JKS. However, this list can change based on the provider in the `java.security` file.

When you specify any of the parameters that begin with `-importSigning`, you must specify them all.

-importSigningCertKSPassword *keystore_password*

Specifies the password of the keystore file that you specify on the `-importSigningCertKS` parameter.

When you specify any of the parameters that begin with `-importSigning`, you must specify them all.

-importSigningCertKSAlias *keystore_alias*

Specifies the alias of the certificate that is in the keystore file that you specify on the `-importSigningCertKS` parameter. The certificate is added to the server default root keystore and is used as the server default root certificate.

When you specify any of the parameters that begin with `-importSigning`, you must specify them all.

-isDefault

Specifies that the profile identified by the accompanying `-profileName` parameter is to be the default profile once it is registered. When issuing commands that address the default profile, it is not necessary to use the `-profileName` attribute of the command.

-isDeveloperServer

Specifies whether the server is intended for development purposes only. This parameter is useful when creating profiles to test applications on a non-production server prior to deploying the applications on their production application servers. The parameter is valid only for the default profile template.

-keyStorePassword *keystore_password*

Specifies the password to use on all keystore files created during profile creation. Keystore files are created for the default personal certificate and the root signing certificate.

-listAugments

Lists the registered augments on a profile that is in the profile registry. You must specify the `-profileName` parameter with the `-listAugments` parameter.

-nodeName *node_name*

Specifies the node name for the node that is created with the new profile. Use a unique value on the machine. Each profile that shares the same set of product binaries must have a unique node name.

The following default node names exist:

- management template for the deployment manager, administrative agent, or the job manager:
profileNameManager
- default template: *shorthostname_profileName*
- managed template: *shorthostname_profileName*
- cell: See the previous management and default template examples and apply as appropriate to the two profiles that are created.
- secureproxy template: *shorthostname_profileName*

The value for this parameter must not contain spaces or any characters that are not valid such as the following: `*`, `?`, `"`, `<`, `>`, `,`, `/`, `\`, `|`, and so on.

-omitAction *feature1 feature2... featureN*

An optional parameter that excludes profile features.

Each profile template comes predefined with certain optional features. The `samplesInstallAndConfig` option is only available when WebSphere Application Server is installed with Sample applications selected. The following optional features can be used with the `-omitAction` parameter for the following profile templates:

- **default** - Application server
 - `deployAdminConsole`
 - `samplesInstallAndConfig`
 - `defaultAppDeployAndConfig`

-personalCertDN *distinguished_name*

Specifies the distinguished name of the personal certificate that you are creating when you create the profile. Specify the distinguished name in quotes. This default personal certificate is located in the server keystore file. The `-importPersonalCertKSType` parameter is mutually exclusive with the `-personalCertDN` parameter. See the `-personalCertValidityPeriod` parameter and the `-keyStorePassword` parameter.

-portsFile *file_path*

An optional parameter that specifies the path to a file that defines port settings for the new profile.

Do not use this parameter when using the `-startingPort` or `-defaultPorts` parameter.

During profile creation, the **manageprofiles** command uses an automatically generated set of recommended ports if you do not specify the `-startingPort` parameter, the `-defaultPorts` parameter or the `-portsFile` parameter. The recommended port values can be different than the default port values based on the availability of the default ports.

-profileName *profile_name*

Specifies the name of the profile. Use a unique value when creating a profile. Each profile that shares the same set of product binaries must have a unique name. The default profile name is based on the profile type and a trailing number, for example:

profileType ProfileNumber

where *profileType* is a value such as `AppSrv`, `Dmgr`, `AdminAgent`, `JobMgr`, or `Custom` and *ProfileNumber* is a sequential number that creates a unique profile name.

The value for this parameter must not contain spaces or characters that are not valid such as the following: `*`, `?`, `"`, `<`, `>`, `,`, `/`, `\`, `|`, and so on. The profile name that you choose must not be in use.

-profilePath *profile_root*

Specifies the fully qualified path to the profile, which is referred to throughout the information center as the *profile_root*.

The default value is based on the *app_server_root* directory, the profiles subdirectory, and the name of the file.

The default value is based on the *user_data_root* directory, the profiles subdirectory, and the name of the file.

For example, the default is:

`WS_WSPROFILE_DEFAULT_PROFILE_HOME/profileName`

The `WS_WSPROFILE_DEFAULT_PROFILE_HOME` element is defined in the `wasprofile.properties` file in the *app_server_root/properties* directory.

The value for this parameter must be a valid path for the target system and must not be currently in use.

The `QEJBSVR` profile must have permissions to write to the directory.

-response *reponse_file*

Accesses all API functions from the command line using the **manageprofiles** command.

The command line interface can be driven by a response file that contains the input arguments for a given command in the properties file in key and value format. Use the following example response file to run a create operation:

```
create
profileName=testResponseFileCreate
profilePath=profile_root
templatePath=app_server_root/profileTemplates/default
nodeName=myNodeName
cellName=myCellName
hostName=myHostName
omitAction=myOptionalAction1, myOptionalAction2
```

To determine which input arguments are required for the various types of profile templates and action, use the `manageprofiles` command with the `-help` parameter.

-restoreProfile

Restores a profile backup. Must be used with the `-backupFile` parameter, for example:

```
manageprofiles(.bat)(.sh) -restoreProfile -backupFile file_name
```

To restore a profile, perform the following steps:

1. Stop the server and the running processes for the profile that you want to restore.
2. Manually delete the directory for the profile from the file system.
3. Run the `-validateAndUpdateRegistry` option of the **manageprofiles** command.
4. Restore the profile by using the `-restoreProfile` option of the **manageprofiles** command.

-samplesPassword *samplesPassword*

Creates a password to be used for samples. The password is used to restrict access to Web application samples installed during the install of the application server.

-serverName *server_name*

Specifies the name of the server. Specify this parameter only for the default and secureproxy templates. If you do not specify this parameter when using the default or secureproxy templates, the default server name is `server1` for the default profile, and `proxy1` for the secure proxy profile.

-serverType `ADMIN_AGENT`

Specifies the type of management profile. Specify `ADMIN_AGENT` for an administrative agent server. This parameter is required when you create a management profile.

-setDefaultName

Sets the default profile to one of the existing profiles. Must be used with the `-profileName` parameter, for example:

```
manageprofiles(.bat)(.sh) -setDefaultName -profileName profile_name
```

-signingCertDN *distinguished_name*

Specifies the distinguished name of the root signing certificate that you create when you create the profile. Specify the distinguished name in quotes. This default personal certificate is located in the server keystore file. The `-importSigningCertKS` parameter is mutually exclusive with the `-signingCertDN` parameter. If you do not specifically create or import a root signing certificate, one is created by default. See the `-signingCertValidityPeriod` parameter and the `-keyStorePassword`.

-signingCertValidityPeriod *validity_period*

An optional parameter that specifies the amount of time in years that the root signing certificate is valid. If you do not specify this parameter with the `-signingCertDN` parameter, the root signing certificate is valid for 20 years.

-startingPort *startingPort*

Specifies the starting port number for generating and assigning all ports for the profile.

Port values are assigned sequentially from the `-startingPort` value.

Do not use this parameter with the `-defaultPorts` or `-portsFile` parameters.

During profile creation, the **manageprofiles** command uses an automatically generated set of recommended ports if you do not specify the `-startingPort` parameter, the `-defaultPorts` parameter or the `-portsFile` parameter. The recommended port values can be different than the default port values based on the availability of the default ports.

Note: Do not use this parameter if you are using the managed profile template.

-templatePath *template_path*

Specifies the directory path to the template files in the installation root directory. Within the `profileTemplates` directory are various directories that correspond to different profile types and that vary with the type of product installed. The profile directories are the paths that you indicate while using the `-templatePath` option. You can specify profile templates that lie outside the installation root, if you happen to have any.

The default template path is `app_server_root/profileTemplates/default`. You can use a relative path for the `-templatePath` parameter. The path is relative to the current working directory or to `app_server_root/profileTemplates`, in that order. The following example creates a profile based on the default for a stand-alone application server:

```
manageprofiles -create -profileName MyProfile -startingPort 10380
```

-unaugment

Augmentation is the ability to change an existing profile with an augmentation template. To unaugment a profile that has been augmented, you must specify the `-unaugment` parameter and the `-profileName` parameter. If a series of `manageprofiles` augmentations were performed, and you specify only these two parameters to unaugment a profile, the `unaugment` action undoes the last `augment` action first.

To unaugment a particular profile that has been augmented, additionally specify the `-ignoreStack` parameter with the `-templatePath` parameter. Normally, you would not unaugment a particular profile because you must ensure that you are not violating profile template dependencies.

When using the `-templatePath` parameter, you can specify a relative file path for the parameter.

See also the `augment` parameter.

-unaugmentAll

Unaugments all profiles that have been augmented with a specific augmentation template. The `-templatePath` parameter is required with the `-unaugmentAll` parameter.

When using the `-templatePath` parameter, you can specify a relative file path for the parameter.

Optionally specify the `-unaugmentDependents` parameter with the `-unaugmentAll` parameter to unaugment all profiles that are prerequisites of the profiles that are being unaugmented.

See also the `augment` parameter.

-unaugmentDependents true | false

If set to `true`, the parameter unaugments all the augmented profiles that are prerequisites to the profiles being unaugmented with the `-unaugmentAll` parameter. The default value for this parameter is `false`.

Optionally specify the `-unaugmentDependents` parameter with the `-unaugmentAll` parameter.

-validateAndUpdateRegistry

Checks all of the profiles that are listed in the profile registry to see if the profiles are present on the file system. Removes any missing profiles from the registry. Returns a list of the missing profiles that were deleted from the registry.

-validateRegistry

Checks all of the profiles that are listed in the profile registry to see if the profiles are present on the file system. Returns a list of missing profiles.

-validatePorts

Specifies the ports that should be validated to ensure they are not reserved or in use. This parameter

helps you to identify ports that are not being used. If a port is determined to be in use, the profile creation stops and an error message displays. You can use this parameter at any time on the create command line. It is recommended to use this parameter with the `-portsFile` parameter.

-webServerCheck true | false

Indicates if you want to set up Web server definitions. Valid values include `true` or `false`. The default value for this parameter is `false`.

-webServerHostname *webserver_host_name*

The host name of the server. The default value for this parameter is the long host name of the local machine.

-webServerInstallPath *webserver_installpath_name*

The installation path of the Web server, local or remote. The default value for this parameter is dependent on the operating system of the local machine and the value of the `webServerType` parameter. For example:

-webServerName *webserver_name*

The name of the Web server. The default value for this parameter is `webserver1`.

-webServerOS *webserver_operating_system*

The operating system from where the Web server resides. Valid values include: `windows`, `linux`, `solaris`, `aix`, `hpux`, `os390`, and `os400`. Use this parameter with the `webServerType` parameter.

-webServerPluginPath *webserver_pluginpath*

The path to the plug-ins that the Web server uses. The default value for this parameter is `WAS_HOME/plugins`.

-webServerPort *webserver_port*

Indicates the port from where the Web server will be accessed. The default value for this parameter is `80`.

-webServerType *webserver_type*

The type of the Web server. Valid values include: `IHS`, `SUNJAVASYSTEM`, `IIS`, `DOMINO`, `APACHE`, and `HTTPSERVER_ZOS`. Use this parameter with the `webServerOS` parameter.

Usage scenario

The following examples demonstrate correct syntax. Issue the command in any of the following examples on one line. Each example shows the command on more than one line to increase clarity.

Creating an administrative agent

The following example uses the management template to create an administrative agent named `AdminAgent001`. The administrative agent ports start at port number `24000`.

```
manageprofiles -create
  -profileName AdminAgent001
  -profilePath profile_root
  -templatePath management
  -serverType ADMIN_AGENT
  -startingPort 24000
```

Creating an application server profile

Create an application server profile named `Default01` with the following command.

```
manageprofiles -create
  -profileName Default01
  -templatePath default
```

```

-startingPort 21000
-personalCertDN "cn=testa, ou=Rochester, o=IBM, c=US"
-signingCertDN "cn=testc, ou=Rochester, o=IBM, c=US"
-keyStorePassword ap3n9krw

```

Logs

The **manageprofiles** command creates a log for every profile that it creates.

- The logs are in the *user_data_root/profileRegistry/logs/manageprofiles* directory. The files are named in this pattern: *profile_name_create.log*.
- The command also creates a log for every profile that it deletes. The logs are in the *user_data_root/profileRegistry/logs/manageprofiles* directory. The files are named in this pattern: *profile_name_delete.log*.

Example: Using predefined port numbers

The **manageprofiles** command recommends initial port values when you do not explicitly set port values. You can use predefined port values instead.

The **manageprofiles** command recommends port values when the options of `-defaultPorts`, `-startingPort`, or `-portsFile` are not specified. The default port values are located in the following files:

Profile	File path
Application server	<i>app_server_root/profileTemplates/default/actions/portsUpdate/portdef.props</i>
Management profile with an administrative agent server	<i>app_server_root/profileTemplates/management/actions/portsUpdate/adminagent.portdef.props</i>

To customize the port values in the *portdef.props* file before creating your profile, perform the following steps. The following example creates the default profile. For other types of profiles, you must substitute the file path with the file path of the profile that you want to create.

1. Copy the *app_server_root/profileTemplates/default/actions/portsUpdate/portdef.props* file from the default profile template path and place a copy of the file in an arbitrary temporary directory such as:
 - `/temp/ports`
2. In the new file, modify the port settings to specify your port values.
3. Create your profile with the **manageprofiles** command. Use the modified port values. Specify the location of your modified *portdef.props* file on the `-portsFile` parameter. Specify the `-validatePorts` parameter to ensure that ports are not reserved or in use. Use the following example as a guide:

```

manageprofiles
  -create
  -profileName Wow_Profile
  -profilePath profile_root
  -templatePath app_server_root\profileTemplates\default
  -nodeName Wow_node
  -cellName Wow_cell
  -hostName lorriemb
  -portsFile \temp\ports\portdef.props
  -validatePorts

```

Suppose that the *portdef.props* file has the following values:

```

WC_defaulthost=39080
WC_adminhost=39060
WC_defaulthost_secure=39443
WC_adminhost_secure=39043
BOOTSTRAP_ADDRESS=32809
SOAP_CONNECTOR_ADDRESS=38880

```

```
IPC_CONNECTOR_ADDRESS=39633
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS=39401
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS=39403
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS=39402
ORB_LISTENER_ADDRESS=39100
DCS_UNICAST_ADDRESS=39353
SIB_ENDPOINT_ADDRESS=37276
SIB_ENDPOINT_SECURE_ADDRESS=37286
SIB_MQ_ENDPOINT_ADDRESS=35558
SIB_MQ_ENDPOINT_SECURE_ADDRESS=35578
SIP_DEFAULTHOST=35060
SIP_DEFAULTHOST_SECURE=35061
```

After running the `manageprofiles` command to create your profile with the user defined port values, a success or fail result displays.

The `manageprofiles` command creates a copy of the current `portdefs.props` file in the `profile_root\properties` directory.

Use only one of the three port values parameters, `-startingPort`, `-defaultPorts`, or `-portsFile` with the `manageprofiles` command. The three parameters are mutually exclusive.

Example: Incrementing default port numbers from a starting point

The `manageprofiles` command can assign port numbers based on a starting port value. You can provide the starting port value from the command line, using the `-startingPort` parameter. The command assigns port numbers sequentially from the starting port number value. However, if a port value in the sequence conflicts with an existing port assignment, the next available port value is used.

The order of port assignments is arbitrary. Predicting assignments is not possible.

For example, ports created with `-startingPort 20002` would appear similar to the following example:

Assigned ports for an Application Server profile

```
WC_defaulthost=20002
WC_adminhost=20003
WC_defaulthost_secure=20004
WC_adminhost_secure=20005
BOOTSTRAP_ADDRESS=20006
SOAP_CONNECTOR_ADDRESS=20007
IPC_CONNECTOR_ADDRESS=20008
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS=20009
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS=20010
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS=20011
ORB_LISTENER_ADDRESS=20012
CELL_DISCOVERY_ADDRESS=20013
NODE_MULTICAST_DISCOVERY_ADDRESS=20014
NODE_IPV6_MULTICAST_DISCOVERY_ADDRESS=20015
NODE_DISCOVERY_ADDRESS=20016
DCS_UNICAST_ADDRESS=20017
SIB_ENDPOINT_ADDRESS=20018
SIB_ENDPOINT_SECURE_ADDRESS=20019
SIB_MQ_ENDPOINT_ADDRESS=20020
SIB_MQ_ENDPOINT_SECURE_ADDRESS=20021
SIP_DEFAULTHOST=20022
SIP_DEFAULTHOST_SECURE=20023
```

Assigned ports for a management profile with an administrative agent server

```
WC_adminhost=20002
WC_adminhost_secure=20003
BOOTSTRAP_ADDRESS=20004
SOAP_CONNECTOR_ADDRESS=20005
IPC_CONNECTOR_ADDRESS=20006
```

```
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS=20007
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS=20008
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS=20009
ORB_LISTENER_ADDRESS=20010
```

The ports are for a DMZ Secure Proxy Server for IBM WebSphere Application Server installation.

```
SOAP_CONNECTOR_ADDRESS=20002
IPC_CONNECTOR_ADDRESS=20003

PROXY_HTTP_ADDRESS=20002
PROXY_HTTPS_ADDRESS=20003
PROXY_SIP_ADDRESS=20004
PROXY_SIPS_ADDRESS=20005
IPC_CONNECTOR_ADDRESS=20006
```

The following example uses the `startingPort` parameter of the `manageprofiles` command and creates ports from an initial value of 20002, with the content shown in the previous example:

```
app_server_root/bin/manageprofiles -create
    -profileName shasti
    -profilePath user_data_root/profiles/shasti
    -templatePath app_server_root/profileTemplates/default
    -nodeName W2K03
    -cellName W2K03_Cell01
    -hostName planetnt
    -startingPort 20002
```

startServer command

The `startServer` command reads the configuration file for the specified server process and starts that server process.

The server process can be an application server or an administrative agent.

You do not have to use a user name and password with the `startServer` command because this command launches a server process but does not invoke an MBean method.

Note: You can use the administrative console to change the Java virtual machine Classpath setting or the environment entries settings for a server. However, before making these changes you should understand the following consequences of making these changes:

- If you change the value of the Java virtual machine Classpath setting, then this new value overrides the value of the Classpath parameter in the launch command that is set, by default, in the script that is generated when the you issue the `startServer -script` command.
- If you add a new environment entry on the Environment entries page or change the setting of an existing entry, then the new and changed values appear as parameters in the script that is generated when you issue the `startServer -script` command.
- If one of the environment entries you add is called PATH, then the value specified for this entry overrides the value specified for the PATH variable that, by default, is set to WAS_PATH in the `setUpCmdLine` file. If the value of the PATH variable is overridden, the following message is sent to the file where your error messages are logged:

```
WSVR0009E: Error occurred during startup. com.ibm.ws.exception.RuntimeError:
java.lang.NoClassDefFoundError: com/ibm/ws/process/Win32ProcessGlue
```

For more information about where to run this command, see the *Using command line tool* topic.

Syntax

The command syntax is one of the following:

```
startServer server_name [options]
```

where *server_name* is the name of the application server that you want to start.

This argument is optional. If this argument is not specified, a server is assigned based on your profile name. If you use the profile named `default`, the `<server>` argument is set to `server1`. If the profile name is not `default`, the `<server>` argument is set to the profile name.

```
startServer <adminagent_name>
```

where *adminagent_name* is the name of the administrative agent that you want to start.

Parameters

The following options are available for the **startServer** command:

-? Prints a usage statement.

-curlib <product_library>

Specifies the current library to use for the underlying Submit Job (SBMJOB) CL command.

-cpyenvvar

Tells the **startServer** command to set the environment variables that are currently defined for the server process. The default is to not set the currently defined environment variables.

-help

Prints a usage statement.

-inllibl <library_list>

Specifies the initial library list to use for the underlying Submit Job (SBMJOB) CL command.

-inlasgrp <ASP_group>

Specifies the initial ASP group for the underlying Submit Job (SBMJOB) CL command.

-jobd <product_library/job_description>

Specifies the job description for the underlying Submit Job (SBMJOB) CL command.

-jobq <product_library/job_queue>

Specifies the job queue for the underlying Submit Job (SBMJOB) CL command.

-nowait

Tells the **startServer** command not to wait for successful initialization of the launched server process.

-outq <product_library/output_queue>

Specifies the output queue for the underlying Submit Job (SBMJOB) CL command.

-profileName

Defines the profile of the server process in a multi-profile installation. The `-profileName` option is not required for running in a single profile environment. The default for this option is the default profile.

-recovery

Specifies that the server will start in recovery mode, perform a transactional recovery, and shut down. The server will not accept any new transactions while it is in recovery mode. When you start the server again, resources that were unavailable due to questionable transactions will be available.

Use this option if a server fails and you do not want to accept new transactions during the recovery process.

Note: If you are migrating from a previous version of the product, make sure that the ENV parameter included on the JCL procedure statement for the controller includes either the `REC=N` or the `REC=Y` element. If the ENV parameter does not include either the `REC=N` or the `REC=Y` element, the server will not restart in recovery mode even if you specify the `-recovery` option.

```
//BB06ACR PROC ENV=,PARMS=' ',REC=N,Z=BB06ACRZ
```

|

-sbs <product_library/subsystem_description>

Specifies the subsystem to use for the underlying Submit Job (SBMJOB) CL command.

-trace

Enables tracing of the native process code that starts the server. The trace output is written to the *was_jobname-jobuser-jobnum.log* file which is located in the *profile_root/logs* directory.

-timeout <seconds>

Specifies the waiting time before server initialization times out and returns an error.

-usejobd

Specifies to use the job description from the Submit Job (SBMJOB) process for the server when switching to the Run-As user ID.

Note: This command does not affect the USER field of the current job description.

Usage scenario

The following examples demonstrate correct syntax. The information within the parentheses is a description of the output that is created if you issue the preceding command.

```
startServer server1 (starts the server1 server for the default profile)
```

```
startServer server1 -trace (starts the server1 server for  
the default profile and produces trace files under the  
profile_root/logs directory)
```

```
startServer -profileName mytest (starts server mytest  
configured under profile mytest)
```

```
startServer AdminAgent01
```

```
startServer.sh server1 -trace -username MyUserName -password MyUserPassword  
-profileName MyProfileName (starts the server1 server using the  
MyProfileName profile. The server runs under the user name MyUserName,  
and produces trace files under the profile_root/logs directory)
```

stopServer command

The stopServer command reads the configuration file for the specified server process. This command sends a Java management extensions (JMX) command to the server telling it to shut down.

The server process can be an application server or an administrative agent server.

By default, the stopServer command does not return control to the command line until the server completes the shut down process. There is a -nowait option to return immediately, as well as other options to control the behavior of the stopServer command. For more information about where to run this command, see the *Using command line tools* topic.

Although the stopServer command returns control when the server completes shut down, it can return before the i5/OS process has terminated. Depending upon the amount of memory which the Java virtual machine (JVM) must return to the memory pool, the process may take several seconds or minutes to completely end. You should not attempt to start the server again until the process has terminated. Use the WRKACTJOB SBS(subsystem) CL command to determine if the process has terminated completely. By default, the subsystem is QWAS7.

Syntax

The command syntax is one of the following:

```
stopServer <server_name> [options]
```


where *server_name* is the name of the configuration directory of the application server or the DMZ Secure Proxy Server for IBM WebSphere Application Server that you want to stop.

This argument is optional. If the default profile is being used, the server defaults to server1. If the profile name is not default, the server defaults to the profile name.

```
stopServer <adminagent_name>
```

where *adminagent_name* is the name of the administrative agent that you want to stop.

Parameters

The following options are available for the stopServer command:

-nowait

Tells the **stopServer** command not to wait for successful shutdown of the server process.

-quiet

Suppresses the progress information that the **stopServer** command prints in normal mode.

-logfile <fileName>

Specifies the location of the log file to which trace information is written. By default, the log file is named stopServer.log and is created in the logs directory.

-profileName

Defines the profile of the server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment. The default for this option is the default profile.

-replacelog

Replaces the log file instead of appending to the current log.

-trace

Generates trace information into a file for debugging purposes. The trace output is written to the stopServer.log file which is located in the *profile_root/logs/server* directory.

-timeout <seconds>

Specifies the time to wait for server shutdown before timing out and returning an error.

-statusport <portNumber>

An optional parameter that allows an administrator to set the port number for server status callback. The tool opens this port and waits for status callback from the server just before the server has completely stopped. If the parameter is not set, an unused port is automatically allocated.

-port <portNumber>

Specifies the server JMX port to use explicitly, so that you can avoid reading the configuration files to obtain the information.

-username <name>

Specifies the user name for authentication if security is enabled in the server. Acts the same as the -user option.

-user <name>

Specifies the user name for authentication if security is enabled in the server. Acts the same as the -username option.

-password <password>

Specifies the password for authentication if security is enabled in the server.

Note: If you are running in a secure environment but have not provided a user ID and password, you will receive the following error message:

```
ADMN0022E: Access denied for the stop operation on Server MBean due
to insufficient or empty credentials.
```

To solve this problem, provide the user ID and password information.

-help

Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
stopServer server1 (stops the server1 server for the default profile)
```

```
stopServer server1 -nowait
```

```
stopServer server1 -trace (produces the stopserver.log file)
```

```
stopServer -profileName mytest (stops server mytest  
configured for profile mytest)
```

```
stopServer AdminAgent01
```

stopNode command

The **stopNode** command reads the configuration file for the Network Deployment node agent process and sends a Java Management Extensions (JMX) command telling the node agent to shut down.

By default, the **stopNode** command waits for the node agent to complete shutdown before it returns control to the command line. There is a **-nowait** option to return immediately, as well as other options to control the behavior of the **stopNode** command. For more information about where to run this command, see the Using command tools article.

Although the **stopServer** command returns control when the server completes shut down, it can return before the i5/OS process has terminated. Depending upon the amount of memory which the Java Virtual Machine must return to the memory pool, the process may take several seconds or minutes to completely end. You should not attempt to start the server again until the process has terminated. Use the WRKACTJOB SBS(subsystem) CL command to determine if the process has terminated completely. By default, subsystem is QWAS7.

Syntax

The command syntax is as follows:

```
stopNode [options]
```

Parameters

The following options are available for the **stopNode** command:

-nowait

Tells the **stopNode** command not to wait for successful shutdown of the node agent process.

-quiet

Suppresses the progress information that the **stopNode** command prints in normal mode.

-logfile <fileName>

Specifies the location of the log file to which trace information is written. By default, the log file is named stopServer.log and is created in the logs directory of the profile for the node being stopped.

-profileName

Defines the profile of the Application Server process in a multi-profile installation. The **-profileName** option is not required for running in a single profile environment. The default for this option is the default profile.

-replacelog

Replaces the log file instead of appending to the current log.

-trace

Generates trace information into a file for debugging purposes. The trace output is written to the `was_jobname-jobuser-jobnum.log` file which is located in the `profile_root/logs` directory.

-timeout <seconds>

Specifies the waiting time for the agent to shut down before timing out and returning an error.

-statusport <portNumber>

An optional parameter that allows an administrator to set the port number for node agent status callback. The tool opens this port and waits for status callback from the node agent just before the node agent has completely stopped. If the parameter is not set, an unused port is automatically allocated.

-stopservers [-saveNodeState]

Stops all application servers on the node before stopping the node agent.

Use the `-saveNodeState` option to save the server restart state so that you can start the application servers based on the previous state using the **startNode** command.

-port <portNumber>

Specifies the node agent JMX port to use explicitly, so that you can avoid reading configuration files to obtain the information.

-username <name>

Specifies the user name for authentication if security is enabled in the node agent. Acts the same as the `-user` option.

-user <name>

Specifies the user name for authentication if security is enabled in the node agent. Acts the same as the `-username` option.

-password <password>

Specifies the password for authentication if security is enabled in the node agent.

Note: If you are running in a secure environment but have not provided a user ID and password, you receive the following error message:

```
ADMN0022E: Access denied for the stop operation on Server MBean due
to insufficient or empty credentials.
```

To solve this problem, provide the user ID and password information.

-help

Prints a usage statement.

Note: When requesting help for the usage statement for the **stopNode** command, a reference to the **stopServer** command displays. All of the options displayed for this usage statement apply to the **stopNode** command.

-? Prints a usage statement.

Note: When requesting help for the usage statement for the **stopNode** command, a reference to the **stopServer** command displays. All of the options displayed for this usage statement apply to the **stopNode** command.

Usage scenario

The following examples demonstrate correct syntax:

stopNode

stopNode -profileName mynode

stopNode -nowait

stopNode -trace (produces trace files under the *profile_root/logs* directory.)

serverStatus command

Use the **serverStatus** command to obtain the status of one or all of the servers configured on a node.

For more information about where to run this command, see the *Using command line tools* topic.

Syntax

The command syntax is as follows:

```
serverStatus <server>|-all [options]
```

The first argument is required. The argument is either the name of the server for which status is desired, or the **-all** keyword which requests status for all servers defined on the node.

Parameters

The following options are available for the **serverStatus** command:

-quiet

Suppresses the progress information that the **serverStatus** command prints in normal mode.

-logfile <fileName>

Specifies the location of the log file to which trace information is written. By default, the log file is named *serverStatus.log* and is created in your *logs* directory.

-profileName

Defines the profile of the Application Server process in a multi-profile installation. The **-profileName** option is not required for running in a single profile environment. The default for this option is the default profile.

-replacelog

Replaces the log file instead of appending to the current log.

-trace

Generates trace information into a file for debugging purposes.

-username <name>

Specifies the user name for authentication if security is enabled. Acts the same as the **-user** option.

-user <name>

Specifies the user name for authentication if security is enabled. Acts the same as the **-username** option.

-password <password>

Specifies the password for authentication if security is enabled.

-help

Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
serverStatus server1  
serverStatus -all (returns status for all defined servers)  
serverStatus -trace (produces the serverStatus.log file)
```

cleanupNode command

The **cleanupNode** command cleans up a node configuration from the cell repository.

Only use this command to clean up a node if you have a node defined in the cell configuration, but the node no longer exists. For more information about where to run this command, see the Using command tools article.

Syntax

The command syntax is as follows:

```
cleanupNode <node name> <deploymgr host> <deploymgr port> [options]
```

where the first argument is required.

Parameters

The following options are available for the **cleanupNode** command:

-quiet

Suppresses the progress information that the **cleanupNode** command prints in normal mode.

-trace

Generates trace information into a log file for debugging purposes.

-profileName

Specifies the deployment manager profile to run the command against. This parameter must be specified if the default profile is not a deployment manager profile.

Usage scenario

The following examples demonstrate correct syntax:

```
cleanupNode myNode -profileName dmgr  
cleanupNode myNode -trace -profileName mydmgr
```

backupConfig command

The backupConfig command is a simple utility to back up the configuration of your node to a file.

By default, all servers on the node stop before the backup is made so that partially synchronized information is not saved. For more information about where to run this command, see the Using command line tools article. If you do not have root authority, you must specify a path for the backup file in a location where you have write permission. The backup file will be in zip format and a .zip extension is recommended.

In a UNIX or Linux environment, the backupConfig command does not save file permissions or ownership information. The restoreConfig command uses the current umask and effective user ID (EUID) to set the permissions and ownership when restoring a file. If it is required that the restored files have the original permissions and ownership, use the tar command (available on all UNIX or Linux systems) to back up and restore the configuration.

The backupConfig command does not save authorities that were granted to the configuration directory structure of the profile. The restoreConfig command sets the owner of the directory structure and its contents to QEJBSVR and restores private authorities to the QTMHHTTP and QNOTES user profiles (if they exist). It does not restore any other private authorities that were granted.

Location

Issue the command from the *profile_root/bin* directory.

Syntax

The command syntax is as follows:

```
backupConfig backup_file [options]
```

where *backup_file* specifies the file to which the backup is written. If you do not specify one, a unique name is generated.

The QEJBSVR user profile must have *WX authority to the directory path specified in *backup_file*. If no path is specified, the QEJBSVR user profile must have *WX authority to the current working directory.

Parameters

The following options are available for the backupConfig command:

-nostop

Tells the backupConfig command not to stop the servers before backing up the configuration

-quiet

Suppresses the progress information that the backupConfig command prints in normal mode

-logfile *file_name*

Specifies the location of the log file to which trace information is written

By default, the log file is named backupConfig.log and is created in the logs directory.

-profileName *profile_name*

Defines the profile of the application server process in a multi-profile installation

The -profileName option is not required for running in a single-profile environment. The default for this option is the default profile.

-replacelog

Replaces the log file instead of appending to the current log

-trace

Generates trace information into the log file for debugging purposes

-username *user_name*

Specifies the user name for authentication if security is enabled in the server; acts the same as the -user option

-user *user_name*

Specifies the user name for authentication if security is enabled in the server; acts the same as the -username option

-password *password*

Specifies the password for authentication if security is enabled in the server

-help

Prints a usage statement

-? Prints a usage statement

Usage

The following example creates a new file that includes the current date:

```
backupConfig WebSphereConfig_2005-04-22.zip
```

The following example creates a file called myprofileBackup.zip under the /home/mydir directory for the myprofile profile.

```
backupConfig /home/mydir/myprofileBackup.zip -profileName myprofile
```

The following example creates a file called myBackup.zip and does not stop any servers before beginning the backup process:

```
backupConfig myBackup.zip -nostop
```

restoreConfig command

Use the restoreConfig command to restore the configuration of your node after backing up the configuration using the backupConfig command.

The restoreConfig command is a simple utility to restore the configuration of your node after backing up the configuration using the backupConfig command. By default, all servers on the node stop before the configuration restores so that a node synchronization does not occur during the restoration. If the configuration directory already exists, it is renamed before the restoration occurs. For more information about where to run this command, see the Using command line tools article.

If you directly make changes to the application files in the *app_server_root/installedApps* directory, a process known as "hot deployment", but do not make the same changes to the application files in the *app_server_root/config* directory, the changes might be overwritten if you use the restoreConfig command.

The backupConfig command does not save file permissions or ownership information. The restoreConfig command uses the current umask and effective user ID (EUID) to set the permissions and ownership when restoring a file. If it is required that the restored files have the original permissions and ownership, use the tar command (available on all UNIX or Linux systems) to back up and restore the configuration.

The restoreConfig command runs under QEJBSVR user profile to ensure that the QEJBSVR user profile is the owner of the directories and files created. The system sets the *PUBLIC authority to the directories that have been created to *EXCLUDE. Any private authorities that previously exist on the directories and files in the configuration directory are lost. Use the grant WebSphere Application Server authority (grtwasaut) Qshell script or the CHGAUT CL command to set any private authorities that were lost.

Note: The QEJBSVR user profile must have at least *X authority to each directory in the path containing the backup_file and *R authority to the backup_file.

The restoreConfig command sets the owner of the directory structure and its contents to the QEJBSVR user profile, but it does not restore private authorities. If you are using an IBM HTTP Server or Lotus Domino HTTP Server instance with the application server on the same system or partition, and the plugin-cfg.xml file for your application server resides under the *profile_root/config* directory structure, use the following instructions to grant the necessary private authorities to the user profile for IBM HTTP Server or Lotus Domino HTTP Server.

Note: If you are not using an IBM HTTP Server or Lotus Domino HTTP Server on the same system as the profile that was restored, do not complete these steps. Also, do not complete these steps if the plugin-cfg.xml file does not reside under the config directory structure for the profile.

1. Sign on to the system.
2. Start a Qshell session using the STRQSH command.
3. Navigate to the *app_server_root/bin* directory for the application server.

4. Use the following grtwasaut Qshell command to grant execute (x) authority to each directory in the path containing the plugin-cfg.xml file, starting with the config directory:

```
grtwasaut -profileName profile_name -object path -dtaut x -user user
```

where *profile_name* is the name of the profile configuration that was restored, *path* is the directory path to modify relative to the profile root directory, and *user* is either QTMHHTTP (for the IBM HTTP Server) or QNOTES (for the Lotus Domino HTTP Server).

For example, run the following commands if you use the IBM HTTP Server for the iSeries® platform and the plugin-cfg.xml file for your myprofile profile resides in the profile_root/config/cells/MYSYSTEM_myprofile/nodes/MYSYSTEM_myprofile/servers/myHTTPInstance directory:

```
grtwasaut -profileName myprofile -object config/cells/MYSYSTEM_myprofile/nodes/MYSYSTEM_myprofile/servers/myHTTPInstance -dtaut x -user QTMHHTTP
grtwasaut -profileName myprofile -object config/cells/MYSYSTEM_myprofile/nodes/MYSYSTEM_myprofile/servers -dtaut x -user QTMHHTTP
grtwasaut -profileName myprofile -object config/cells/MYSYSTEM_myprofile/nodes/MYSYSTEM_myprofile -dtaut x -user QTMHHTTP
grtwasaut -profileName myprofile -object config/cells/MYSYSTEM_myprofile/nodes -dtaut x -user QTMHHTTP
grtwasaut -profileName myprofile -object config/cells/MYSYSTEM_myprofile -dtaut x -user QTMHHTTP
grtwasaut -profileName myprofile -object config/cells -dtaut x -user QTMHHTTP
grtwasaut -profileName myprofile -object config -dtaut x -user QTMHHTTP
```

Use the following example if you are using IBM HTTP Server for the iSeries platform and the plugin-cfg.xml file for your myprofile profile resides in the *profile_root*/config/cells directory:

```
grtwasaut -profileName myprofile -object config/cells -dtaut x -user QTMHHTTP
grtwasaut -profileName myprofile -object config -dtaut x -user QTMHHTTP
```

5. Use the following grtwasaut command to grant read,execute (rx) authority to the plugin-cfg.xml file:

```
grtwasaut -profileName profile_name -object path/plugin-cfg.xml -dtaut x -user user
```

where *profile_name* is the name of the profile configuration that was restored, *path* is the directory path to modify relative to the profile root directory, and *user* is QTMHHTTP (for the IBM HTTP Server) or QNOTES (for the Lotus Domino HTTP Server).

For example, enter the following command if you use the IBM HTTP Server for the iSeries platform and the plugin-cfg.xml file for your myprofile profile resides in the *profile_root*/config/cells/MYSYSTEM_myprofile/nodes/MYSYSTEM_myprofile/servers/myHTTPInstance directory:

```
grtwasaut -profileName myprofile -object config/cells/MYSYSTEM_myprofile/nodes/MYSYSTEM_myprofile/servers/myHTTPInstance/plugin-cfg.xml -dtaut rx -user QTMHHTTP
```

Use the following example if you are using IBM HTTP Server for the iSeries platform and the plugin-cfg.xml file for your myprofile profile resides in *profile_root*/config/cells directory:

```
grtwasaut -profileName myprofile -object config/cells/plugin-cfg.xml -dtaut rx -user QTMHHTTP
```

Location

Issue the command from the *profile_root*/bin directory.

Syntax

The command syntax is as follows:

```
restoreConfig backup_file [options]
```

where *backup_file* specifies the file to be restored. If you do not specify one, the command will not run.

Parameters

The following options are available for the restoreConfig command:

-help

Prints a usage statement

-location *directory_name*

Specifies the directory where the backup file is restored

The location defaults to the *profile_root*/config directory.

-logfile *file_name*

Specifies the location of the log file to which trace information is written

By default, the log file is named restoreConfig.log and is created in your logs directory.

-nostop

Tells the restoreConfig command not to stop the servers before restoring the configuration

-password *password*

Specifies the password for authentication if security is enabled in the server

-profileName *profile_name*

Defines the profile of the Application Server process in a multiple-profile installation

The -profileName option is not required for running in a single profile environment. The default for this option is the default profile.

-quiet

Suppresses the progress information that the restoreConfig command prints in normal mode

-replacelog

Replaces the log file instead of appending to the current log

-trace

Generates trace information into the log file for debugging purposes

-username *user_name*

Specifies the user name for authentication if security is enabled in the server; acts the same as the -user option

-user *user_name*

Specifies the user name for authentication if security is enabled in the server; acts the same as the -username option

-? Prints a usage statement

Usage

The following example demonstrates correct syntax:

```
restoreConfig WebSphereConfig_2006-04-22.zip
```

The following example restores the given file to the /tmp directory and does not stop any servers before beginning the restoration:

```
restoreConfig WebSphereConfig_2006-04-22.zip -location /tmp -nostop
```

The following example restores the configuration stored in /home/mydir/myprofileBackup.zip to the configuration for profile myprofile:

```
restoreConfig /home/mydir/myprofileBackup.zip -profileName myprofile
```

Be aware that if you restore the configuration to a directory that is different from the directory that was backed up when you performed the backupConfig command, you might need to manually update some of the paths in the configuration directory.

EARExpander command

Use the **EARExpander** command to expand an enterprise archive file (EAR) into a directory to run the application in that EAR file.

You can collapse a directory containing application files into a single EAR file. You can type `EARExpander` with no arguments to learn more about its options. For more information about where to run this command, see the [Using command tools](#) article.

Syntax

The command syntax is as follows:

```
EarExpander -ear earName -operationDir dirName -operation  
<expand | collapse> [-expansionFlags <all|war>]
```

Parameters

The following options are available for the **EARExpander** command:

-ear

Specifies the name of the input EAR file for the expand operation or the name of the output EAR file for the collapse operation.

-operationDir

Specifies the directory where the EAR file is expanded or specifies the directory from where files are collapsed.

-operation <expand | collapse>

The `expand` value expands an EAR file into a directory structure required by the WebSphere Application Server run time. The `collapse` value creates an EAR file from an expanded directory structure.

-expansionFlags <all | war>

(Optional) The `all` value expands all files from all of the modules. The `war` value only expands the files from Web archive file (WAR) modules.

-profileName

Defines the profile of the Application Server process in a multi-profile installation. The `-profileName` option is not required for running in a single profile environment. The default for this option is the default profile.

Usage scenario

The following examples demonstrate correct syntax:

```
EARExpander.sh -ear /WebSphere/AppServer/installableApps/DefaultApplication.ear  
-operationDir /MyApps -operation expand -expansionFlags war
```

```
EARExpander.sh -ear /backup/DefaultApplication.ear  
-operationDir /MyAppsDefaultApplication.ear -operation collapse
```

GenPluginCfg command

The `GenPluginCfg` command is used to regenerate the WebSphere Web server plug-in configuration file, `plugin-cfg.xml`.

For more information about where to run this command, see the [Using command tools](#) article.

Note: Regenerating the plug-in configuration can overwrite manual configuration changes that you might want to preserve. Before performing this task, understand its implications as described in the [Communicating with Web servers](#) topic in the *Setting up the application serving environment* PDF.

Syntax

To regenerate the plug-in configuration perform one of the following:

- Issue the following command:

```
app_server_root/bin/GenPluginCfg
```

This method for regenerating the plug-in configuration creates a `plugin-cfg.xml` file in ASCII format, which is the proper format for execution in a z/OS environment.

You can use the `-profileName` option to define the profile of the application server process in a multi-profile installation. The `-profileName` option is not required for running in a single profile environment. The default for this option is the default profile.

When the `GenPluginCfg` command is issued with the option `-webserver.name webserverName`, `wsadmin` generates a plug-in configuration file for the Web server. The settings in the generated configuration file are based on the list of applications that are deployed on the Web server. When this command is issued without the option `-webserver.name webserverName`, the plug-in configuration file is generated based on topology.

Parameters

The following options are available for the `GenPluginCfg` command:

-config.root configroot_dir

Defaults to `CONFIG_ROOT`. The `setupCmdLine` command is invoked to get this environment variable.

-profileName

Defines the profile of the Application Server process in a multi-profile installation. The `-profileName` option is not required for running in a single profile environment. The default for this option is the default profile.

-cell.name cell

Defaults to `WAS_CELL`. The `setupCmdLine` command is invoked to get this environment variable.

-node.name node

Defaults to `WAS_NODE`. The `setupCmdLine` command is invoked to get this environment variable.

-webserver.name webserver1

Required for creating plug-in configuration file for a given Web server.

-propagate yes/no

Applicable only when the `webserver.name` option is specified and the Web server is local. Otherwise, you must manually copy the `plugin-cfg.xml` file from `app_server_root/profiles/profile_name/config/cells/cell_name/nodes/node_name/servers/web_server_name` to `plugins_root/config/web_server_name` in the remote Web server plugins directory. The default value is `no`.

-propagateKeyring yes/no

Applicable only when the option `webserver.name` is specified and the Web server is local. Defaults to `no`.

-cluster.name cluster1,cluster2 | ALL

Optional list of clusters. Ignored when the option `webserver.name` is specified.

-server.name server1,server2

Optional list of servers. Required for single server plug-in generation. Ignored when the option `webserver.name` is specified.

-output.file.name file_name

Defaults to the `configroot_dir/plugin-cfg.xml` file. Ignored when the option `webserver.name` is specified.

-destination.root root

Installation root of the machine configuration is used on. Ignored when the option `webserver.name` is specified.

-destination.operating.system windows/unix

Operating system of the machine configuration is used on. Ignored when the option `webserver.name` is specified.

-debug yes/no

Defaults to no.

-help

Prints a usage statement.

-? Prints a usage statement.

Usage scenario

To generate a plug-in configuration for all of the clusters in a cell:

To generate a plug-in configuration for a single server:

To generate a plug-in configuration file for a Web server:

versionInfo command

The `versionInfo` command generates a report from data extracted from XML files in the `properties/version` folder. The report includes a list of changed components and installed or uninstalled maintenance packages.

Product version information

The `versionInfo` tool displays important data about the product and its installed components, such as the build version and build date. History information for installation and removal of maintenance packages also displays in the report. This tool is particularly useful when working with support personnel to determine the cause of any problem.

Product version reports

The following report generation scripts extract data from XML data files in the `app_server_root/properties/version` folder:

- `versionInfo` script
Lets you use parameters to create a version report.
- `genVersionReport` script
Generates the `versionReport.html` report file in the current working directory, which is usually the `app_server_root/bin` directory.

Location of the command file

`app_server_root/bin/versionInfo`

Syntax for the versionInfo command

The command syntax is:

```
versionInfo [ -format text | html ]  
            [ -file file_name ]  
            [ -long ]  
            [ -maintenancePackages ]  
            [ -maintenancePackageDetail ]  
            [ -components ]
```

[-componentDetail]

versionInfo [-help | /help | -? | /? | -usage]

Issue the command from the bin directory of *app_server_root*.

Parameters

-? or /?

Displays command syntax.

-components

Adds a list of installed components to the report.

-componentDetail

Adds details about installed components to the report.

-file *file_name*

Specifies the output file name. The report goes to standard output (stdout) by default.

-format text | html

Selects the format of the report. The default is "text".

-help or /help

Displays command syntax.

-long

Creates the long version of the report.

-maintenancePackages

Adds a list of applied maintenance packages to the report.

-maintenancePackageDetail

Adds details about an applied maintenance package to the report.

-usage

Displays command syntax.

Non-root limitations

You cannot run the versioninfo command with a user which does not have authority to write to the main product files. You can avoid the problem by running the versioninfo command with a user with authority to write to *app_server_root/properties/version/history* or by changing the permissions of the directory to allow other users to write to it.

Note: If you are using zones with your Solaris operating system, then you must run the versionInfo command in the global zone.

Report description

The versionInfo command reports the following information:

Installation information: Displays the following general information about the current installation:

- Report date and time - The date and time that the report was generated. The timestamp is formatted according to the current locale.
- Product directory - The file path to the installation root directory defined by the WAS_HOME environment variable.
- Version directory - The file path of the version directory of the current IBM WebSphere Application Server installation.
- DTD directory - The file path of the DTD directory of the current IBM WebSphere Application Server installation.

- Log directory - The file path of the log directory of the current IBM WebSphere Application Server installation. The maintenance package log files are in the directory.
- Backup directory - The file path of the backup directory of the current IBM WebSphere Application Server installation. The backup files generated during the installation of maintenance packages are in this directory.
- TMP directory - The file path of the temporary directory of the current machine.

Installation platform information: Displays information from the platform.websphere file:

- Name - IBM WebSphere Application Server
- Version - 7.0

Product list information: Displays a list of installed WebSphere products:

- Product ID - The product ID of the installed product.
- Status - The status of the product, either installed or uninstalled.

Installed product information: This information and the other information topic descriptions are hierarchal for each installed product, component, component update, installed maintenance package, included APARs, and component updates.

This section of the report displays the following information:

- Name - The name of the installed product.
- Version - The current version of the product. Installing or uninstalling fix packs or refresh packs modifies this version.
- ID - The product ID of the product installed, such as BASE, BASETRIAL, ND, EXPRESS, EXPRESSTRIAL, embeddedEXPRESS, IHS, XD, PLG, or CLIENT.
- Build level - The build level of the installed product.
- Build date - The build date of the installed product.

Installed component information: Displays the following component-level information of the installed component from the .component file under the /properties/version directory:

- Component name - The name of the installed component
- Spec version - The spec version of the current component
- Build level - The build level of the current component
- Build date - The build date of the current component

Installed component update information: Displays the general maintenance package information:

- Component name - the name of the installed component
- Update type - All interim fixes, fix packs and refresh packs are maintenance packages. The update type field is always set to *maintenance package*.
- Maintenance package ID - The ID of the maintenance package that is responsible for the current maintenance level of the product.
- Update effect - The updated action taken on a particular component. The default effect is *replace*.
- Log file name - The file path of the log file that records the maintenance actions for the current maintenance level.
- Backup file name - The file path of the backup file generated during the installation of the maintenance package. Not applicable if the component is restored as a result of an uninstall of a maintenance package.
- Timestamp - The time at which the component is updated. The timestamp is stated in GMT offset values.

Installed maintenance package information: Displays the general maintenance package information:

- Maintenance Package ID - the maintenance package ID
- Description - the description of the maintenance package
- Build Date - the build date of this maintenance package

Included APARs information: Displays the list of APARs fixed by this maintenance package.

Component updates information: Displays the following information about each component that is updated by the installed maintenance package:

- Component name - The name of the installed component.
- Updated effect - The update action taken on the component by the maintenance package. The default action is *replace*.
- Timestamp - The time at which the component update is installed (GMT offset).

genVersionReport command

The `genVersionReport` command uses the `versionInfo` command to generate the `versionReport.html` report file in the current working directory, which is usually the `bin` directory. The report includes a list of changed components and installed or uninstalled maintenance packages.

Product version information

The `versionInfo` tool displays important data about the product and its installed components, such as the build version and build date. History information for installation and removal of maintenance packages also displays in the report. This tool is particularly useful when working with support personnel to determine the cause of any problem.

Product version reports

The following report generation scripts extract data from XML data files in the `properties/version` folder:

- `versionInfo` script
Use the `versionInfo` command to specify your own report parameters when creating a customized version report.
- `genVersionReport` script
Use the `genVersionReport` command to generate the `versionReport.html` report file in the current working directory, which is usually the `bin` directory. The report includes the list of components, fixes, fix packs, and refresh packs.

Location of the command file

`app_server_root/bin/genVersionReport`

Syntax for the genVersionReport command

The command syntax is:

```
genVersionReport
```

Issue the command from the `bin` directory of the `app_server_root` directory.

Report description

The `versionInfo` command reports the following information:

Installation information: Displays the following general information about the current installation:

- Report date and time - The date and time that the report was generated. The timestamp is formatted according to the current locale.
- Product directory - The file path to the installation root directory defined by the `WAS_HOME` environment variable.

- Version directory - The file path of the version directory of the current IBM WebSphere Application Server installation.
- DTD directory - The file path of the DTD directory of the current IBM WebSphere Application Server installation.
- Log directory - The file path of the log directory of the current IBM WebSphere Application Server installation. The maintenance package log files are in the directory.
- Backup directory - The file path of the backup directory of the current IBM WebSphere Application Server installation. The backup files generated during the installation of maintenance packages are in this directory.
- TMP directory - The file path of the system temporary directory.

Installation platform information: Displays information from the platform.websphere file:

- Name - IBM WebSphere Application Server
- Version - 7.0

Product list information: Displays a list of installed WebSphere products:

- Product ID - The product ID of the installed product.
- Status - The status of the product, either installed or uninstalled.

Installed product information: This information and the other information topic descriptions are hierarchal for each installed product, component, component update, installed maintenance package, included APARs, and component updates.

This section of the report displays the following information:

- Name - The name of the installed product.
- Version - The current version of the product. Installing or uninstalling fix packs or refresh packs modifies this version.
- ID - The product ID of the product installed, such as BASE, BASETRIAL, ND, EXPRESS, EXPRESSTRIAL, embeddedEXPRESS, IHS, XD, PLG, or CLIENT.
- Build level - The build level of the installed product.
- Build date - The build date of the installed product.

Installed component information: Displays the following component-level information of the installed component from the .component file in the *app_server_root/properties/version* directory:

- Component name - The name of the installed component
- Spec version - The spec version of the current component
- Build level - The build level of the current component
- Build date - The build date of the current component

Installed component update information: Displays the general maintenance package information:

- Component name - the name of the installed component
- Update type - All interim fixes, fix packs and refresh packs are maintenance packages. The update type field is always set to *maintenance package*.
- Maintenance package ID - The ID of the maintenance package that is responsible for the current maintenance level of the product.
- Update effect - The updated action taken on a particular component. The default effect is *replace*.
- Log file name - The file path of the log file that records the maintenance actions for the current maintenance level.
- Backup file name - The file path of the backup file generated during the installation of the maintenance package. Not applicable if the component is restored as a result of an uninstall of a maintenance package.

- Timestamp - The time at which the component is updated. The timestamp is stated in GMT offset values.

Installed maintenance package information: Displays the general maintenance package information:

- Maintenance Package ID - the maintenance package ID
- Description - the description of the maintenance package
- Build Date - the build date of this maintenance package

Included APARs information: Displays the list of APARs fixed by this maintenance package.

Component updates information: Displays the following information about each component that is updated by the installed maintenance package:

- Component name - The name of the installed component.
- Updated effect - The update action taken on the component by the maintenance package. The default action is *replace*.
- Timestamp - The time at which the component update is installed (GMT offset).

historyInfo command

The **historyInfo** command generates a report from data extracted from XML files in the properties/version folder and the properties/version/history folder. The report includes a list of changed components and a history of installed or uninstalled maintenance packages.

Product history information

The historyInfo tool displays important data about the product and its installed components, such as the build version and build date. History information for installation and removal of maintenance packages also displays in the report. This tool is particularly useful when working with support personnel to determine the cause of any problem.

Product history reports

The following report generation scripts extract data from XML data files in the properties/version folder and the properties/version/history folder:

- historyInfo script
Lets you use parameters to create a history report.
- genHistoryReport script
Generates the historyReport.html report file in the current working directory, which is usually the bin directory.

Location of the command file

app_server_root/bin/historyInfo

Syntax for the historyInfo command

The command syntax is:

```
historyInfo [ -format text | html ]
            [ -file file_name ]
            [ -maintenancePackageID ID_of_maintenance_package ]
            [ -component component_name ]
```

```
historyInfo [ -help | /help | -? | /? | -usage ]
```

Issue the command from the bin directory of the *app_server_root* directory.

Parameters

-? or /?

Displays command syntax.

-component *component_name*

Specifies the name of a component. When specified, the product history report displays events for only the named component. When not specified, the report displays events for all components.

-file *file_name*

Specifies the output file name. The report goes to standard output (stdout) by default.

-format *text | html*

Selects the format of the report. The default is "text".

-help or /help

Displays command syntax.

-maintenancePackageID *ID_of_maintenance_package*

Specifies the ID of the interim fix, fix pack, or refresh pack. When specified, the product history report displays events for only the named maintenance package. When not specified, the report displays events for all maintenance packages.

-usage

Displays command syntax.

Report description

The historyInfo command reports the following information:

Installation information: Displays the following general information about the current installation:

- Report date and time - The date and time that the report was generated. The timestamp is formatted according to the current locale.
- Product directory - The file path to the installation root directory of the product.
- Version directory - The file path of the version directory of the current WAS installation.
- DTD directory - The file path of the DTD directory of the current installation.
- Log directory - The file path of the log directory of the current installation. The maintenance package log files are in the directory.
- Backup directory - The file path of the backup directory of the current WAS installation. The backup files generated during the installation of maintenance packages are in this directory.
- TMP directory - The file path of the temporary directory of the current machine.
- History directory - The file path of the history directory of the current installation. The history files are in the directory.
- History File - The file path of the event.history file.

Installation event information: Displays the list of installed maintenance packages (interim fix, fix pack, and refresh pack) and the following related information:

- Maintenance package ID - The ID of the maintenance package.
- Action - The action taken with this maintenance package, which is either *install* or *uninstall*.
- Package file name - The file name of the maintenance package that was installed.
- Log file name - The file path of the log file generated during the installation or removal of the maintenance package.
- Backup file name - The file path of the backup file generated during the installation of the maintenance package. This field does not apply for an uninstall action.
- Timestamp - The time when the maintenance action (install or uninstall) occurs. The time is stated in relation to GMT.

- Result - The result of the installation or removal action. The result is either success, partial success, or failure.

Component installation event information: Displays the following component-level information of the event for the current maintenance package:

- Maintenance package ID - The ID of the maintenance package to which this particular installation event belongs.
- Component name - The name of the current component.
- Action - The action taken on this component due to the action of the current maintenance package, either install or uninstall
- Update action - The updated action taken on this component. By default, the update action for an installation action is *replace*.
- Timestamp - The time at which the action occurs for the maintenance package (GMT offset values).
- Result - The result of the install or uninstall action. The result is either success, partial success or failure.

genHistoryReport command

The **genHistoryReport** command generates the historyReport.html report file in the current working directory, which is usually the bin directory. The report includes a list of changed components and installed or uninstalled maintenance packages. The genHistoryReport script invokes the historyInfo script specifying the correct parameters to place the information generated into an HTML file in the current directory.

Product history information

The historyInfo tool displays historical data about the product and the installation and removal of maintenance packages for the product. This tool is particularly useful when working with support personnel to determine the cause of any problem.

Product history reports

The following report generation scripts extract data from XML data files in the properties/version folder and the properties/version/history folder:

- historyInfo script
Lets you use parameters to create a history report.
- genHistoryReport script
Generates the historyReport.html report file in the current working directory, which is usually the bin directory. The report includes a list of components, fixes, fix packs, and refresh packs.

Location of the command file

app_server_root/bin/genHistoryReport

Syntax for the genHistoryReport command

The command syntax is:

```
genHistoryReport
```

Issue the command from the bin directory of the *app_server_root* directory.

Report description

The historyInfo command generates the report. The genHistoryReport command calls the historyInfo command with a set of report parameters that reports the following information:

Installation information: Installation information displays the following general information about the current installation:

- Report date and time - The date and time that the report was generated. The timestamp is formatted according to the current locale.
- Product directory - The file path to the installation root directory of the product.
- Version directory - The file path of the version directory of the current WAS installation.
- DTD directory - The file path of the DTD directory of the current installation.
- Log directory - The file path of the log directory of the current installation. The maintenance package log files are in the directory.
- Backup directory - The file path of the backup directory of the current WAS installation. The backup files generated during the installation of maintenance packages are in this directory.
- TMP directory - The file path of the temporary directory of the current machine.
- History directory - The file path of the history directory of the current installation. The history files are in the directory.
- History File - The file path of the event.history file.

Installation event information: Installation event information displays the list of installed maintenance packages (interim fix, fix pack, and refresh pack) and the following related information:

- Maintenance package ID - The ID of the maintenance package.
- Action - The action taken with this maintenance package, which is either *install* or *uninstall*.
- Package file name - The file name of the maintenance package that was installed.
- Log file name - The file path of the log file generated during the installation or removal of the maintenance package.
- Backup file name - The file path of the backup file generated during the installation of the maintenance package. This field does not apply for an uninstall action.
- Timestamp - The time when the maintenance action (install or uninstall) occurs. The time is stated in relation to GMT.
- Result - The result of the installation or removal action. The result is either success, partial success, or failure.

Component installation event information: Component installation event information displays the following component-level information of the event for the current maintenance package:

- Maintenance package ID - The ID of the maintenance package to which this particular installation event belongs.
- Component name - The name of the current component.
- Action - The action taken on this component due to the action of the current maintenance package, either install or uninstall
- Update action - The updated action taken on this component. By default, the update action for an installation action is *replace*.
- Timestamp - The time at which the action occurs for the maintenance package (GMT offset values).
- Result - The result of the install or uninstall action. The result is either success, partial success or failure.

Granting authority to a profile using the i5/OS command line

To grant authority to objects and directories in a profile, run the **grtwasaut** command from the Qshell command line.

About this task

To run the script, perform the following steps:

1. On the i5/OS command line, run the STRQSH (Start Qshell) command.
2. On the Qshell command line, use the cd command to change to the directory that contains the script:

```
cd app_server_root/bin
```

3. Run the **grtwasaut** command:

```
grtwasaut -profileName profile [ -user usrprf | -authlist authlist ]  
-dtaut dataAuth -objaut objAuth
```

where *profile* is the profile to which you are granting authority, *usrprf* is the user profile to which you are granting authority, *authlist* is the authorization list to which you are granting authority, *dataAuth* specifies the data authorities that you are granting to the user specified by the *-user* parameter and *objAuth* specifies the object authorities that you are granting to the user specified by the *-user* parameter. You do not need to specify both the *-user* and *-authlist* parameters, but you must specify at least one of them. For more information on the **grtwasaut** command and additional parameters, see the **grtwasaut** command article. Examples

Example

In the following example, the user profiles *johndoe* and *jsmith* are granted *rxw* authority to the profile *devinst* and the associated objects.

```
grtwasaut -profileName devinst -user "johndoe jsmith" -dtaut rxw -recursive
```

In the following example, the user profiles *johndoe* and *jsmith* are granted *rxw* authority to the *installedApps* subdirectory and all nested objects in the *installedApps* subdirectory.

```
grtwasaut -profileName devinst -object installedApps -user "johndoe jsmith" -dtaut rxw -recursive
```

Revoking authority to a profile using the **i5/OS** command line

To revoke authority to objects and directories in a profile, run the *rvkwasaut* script from the Qshell command line.

About this task

To run the script, follow these steps:

1. On the **i5/OS** command line, run the **STRQSH** (Start Qshell) command.
2. On the Qshell command line, use the **cd** command to change to the directory that contains the script:

```
cd app_server_root/bin
```

3. Run the *rvkwasaut* script:

```
rvkwasaut -profileName profile [-user usrprf | -authlist authlist]
```

where *profile* is the *profile* to which you want are revoking authority, *usrprf* is the user profile from which you are revoking authority, and *authlist* is the authorization list from which you are revoking authority. You do not need to specify both the *-user* and *-authlist* parameters, but you must specify at least one of them. For more information on the *rvkwasaut* script and additional parameters, see the “*rvkwasaut* command” on page 198 article.

Example

In the following example, the user profile *jsmith* no longer has authority to the profile *devinst*:

```
rvkwasaut -profileName devinst -user jsmith -recursive
```

In the example, the user profile *jsmith* no longer has authority to the *installedApps* subdirectory and all nested objects in the *installedApps* subdirectory.

```
rvkwasaut -profileName devinst -object installedApps -user jsmith -recursive
```

checkprereqs command

The **checkprereqs** command runs the prerequisite validator tool.

This tool verifies your WebSphere Application Server installation, ensures that you have the appropriate prerequisite software and PTFs installed, and verifies certain configuration settings that the product requires. To run this script, your user profile must have *ALLOBJ authority.

Syntax

The command syntax is as follows:

```
checkprereqs [-xmlinput xml_input_file] [-xmloutput xml_output_file]  
             [-output output_file] [-trace trace_file] [-help]
```

Parameters

The following options are available for the **checkprereqs** command:

-xmlinput

This is an optional parameter. The value *xml_input_file* specifies the fully qualified path of an XML file that the script uses. If you do not specify this parameter, the script uses the CheckPrereqs.xml file that is included in the ServiceTools.jar file.

-xmloutput

This is an optional parameter. The value *xml_output_file* specifies the fully qualified path of an XML file to which the script writes its results. If you do not specify this parameter, the script writes its output to the display.

-output

This is an optional parameter. The value *output_file* specifies the fully qualified path of a file to which the script writes its standard output. If you do not specify this parameter, the script does not generate standard output.

-trace

This is an optional parameter. The value *trace_file* specifies the fully qualified path of a file to contain trace information. If you do not specify this parameter, the script does not generate trace information.

-verbose

This is an optional parameter. If you specify this parameter, the script displays all result information. If you do not specify this parameter, the script does not display successful result information.

-help

This optional parameter displays the usage statement for the script.

chgwassvr command

The **chgwassvr** command allows you to change the ports for an application server within a profile.

For usage instructions and examples, see the Change application server properties article. To run this script, your user profile must have *ALLOBJ authority.

You can also use the administrative console to create and manage application servers. See the Administering application servers article for more information.

Syntax

The command syntax is as follows:

```
chgwassvr -server servername  
          [instance instancename] [-profileName instancename]  
          [-portblock portnumber] [-transport -oldport oldvalue  
          -newport newvalue] [-endpoint endpointname -port newvalue]  
          [-inhttp inhttpport]  
          [-inhttps inhttpsport] [-admin adminport]  
          [-adminssl adminsslport]
```

```
[-soap soapport] [-nameservice nameserviceport]
[-sas sasserverport]
[-csiv2server csiv2serverauthport]
[-csiv2client csiv2clientauthport] [-verbose] [-help | -?]
```

Parameters

The following options are available for the **chgwassvr** command:

-server

This is a required parameter. The value *servername* specifies the name of the server to change. For the default WebSphere Application Server profile, the default server name is *server1*. For the default Network Deployment profile, the default server name is *dmgr*. The value for this parameter is case-sensitive and must match the server name exactly. You can use the **dspwasinst** command to view the servers associated with a profile. For more information, see the “dspwasinst command” on page 191 article.

-profileName

This is an optional parameter. The value *profile* specifies the name of the profile that contains the application server you want to change. The default value is *default*.

-portblock

This is an optional parameter. The value *portblock* specifies the first number of a block of port numbers that your profile uses. If you specify this parameter, the script changes all of the port numbers for your application server. If you do not specify this parameter, the port numbers for your application server are not changed, unless you specify a port parameter to change (see port parameters below). You can use the Work with TCP/IP Network Status (NETSTAT *CNN) command to display a list of port numbers that are currently being used.

- A WebSphere Application Server profile uses several ports for various functions. When you change an application server’s properties, ports are assigned based on the following ordered conditions:
 - **Specific port parameters** If you specify values for specific port parameters, the script uses those values. Specific port parameters are *-inhttp* (Base application server), *-admin*, *-jmsqueued* (Base application server), *-jmsdirect* (Base application server), *-jmssecure* (Base application server), *-soap*, *-orblister* (Network Deployment), *-nameservice*, *-drsclient*, and *-celldiscovery* (Network Deployment).
 - **The -portblock parameter** Services for which you have not specified a port number are assigned ports sequentially starting with the value of the *-portblock* parameter. If a script encounters a port that is in use, it skips that port number and continues with the next unused port.
 - **Current values** If *-portblock* is not specified, any services for which you have not specified a port parameter retain their current value.
- For the *-portblock* parameter, the script checks only a master index of all profiles of WebSphere Application Server Version 5.1 and WebSphere Application Server Network Deployment Version 5.1. The script is not able to detect port usage by other applications, including previous versions of WebSphere Application Server.

-transport

This is an optional parameter set. The value *oldvalue* specifies the port number of the Web container transport that you want to change. The value *newvalue* specifies the new port number that you want to assign to the Web container transport. If the Web Container transport is also specified in the host alias for a virtual host, the port for the host alias is also updated.

Use the *-transport* parameter set instead of the *-admin*, *-adminssl*, *-inhttp* or *-inhttpssl* parameters. These parameters are deprecated.

-endpoint

This is an optional parameter. The value *endpointname* specifies the name of the endpoint port to modify. The value *newvalue* specifies port number to which to set the endpoint’s port.

-inhttp

This is an optional parameter. The value *inhttpport* specifies the port number on which the Web container listens for requests from the Web server. If neither the *-portblock* parameter nor the *-inhttp* parameter is specified, this port is not changed. See the *-portblock* parameter for more information.

-inhttps

This is an optional parameter. The value *inhttpsport* specifies the port number that your application server uses for secure communications with internal HTTP server. If neither the *-portblock* parameter nor the *-inhttps* parameter is specified, this port is not changed. See the *-portblock* parameter for more information.

-admin

This is an optional parameter. The value *adminport* specifies the port number to use for the administrative console. If neither the *-portblock* parameter nor the *-admin* parameter is specified, this port is not changed. See the *-portblock* parameter for more information.

-adminssl

This is an optional parameter. The value *adminportssl* specifies the port number to use for the secure communications with administrative console. If neither the *-portblock* parameter nor the *-adminssl* parameter is specified, this port is not changed. See the *-portblock* parameter for more information.

-soap

This is an optional parameter. The value *soapport* specifies the port number to use for SOAP. If neither the *-portblock* parameter nor the *-soap* parameter is specified, this port is not changed. See the *-portblock* parameter for more information.

-nameservice

This is an optional parameter. The value *nameserviceport* specifies the port number to use for name service (or RMI connector) port. If neither the *-portblock* parameter nor the *-nameservice* parameter is specified, this port is not changed. See the *-portblock* parameter for more information.

-sas

This is an optional parameter. The value *sasserverport* specifies the port on which the Secure Association Services (SAS) listen for inbound authentication requests. If the *-sas* parameter is not specified, this port is not changed. This port is specified by the *SAS_SSL_SERVERAUTH_LISTENER_ADDRESS* property in *serverindex.xml*.

-csiv2server

This is an optional parameter. The value *csiv2serverauthport* specifies the port on which the Common Secure Interoperability Version 2 (CSIV2) Service listens for inbound server authentication requests. If the *-csiv2server* parameter is not specified, this port is not changed. This port is specified by the *CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS* property in *serverindex.xml*.

-csiv2client

This is an optional parameter. The value *csiv2clientauthport* specifies the port on which the Common Secure Interoperability Version 2 (CSIV2) Service listens for inbound client authentication requests. If the *-csiv2client* parameter is not specified, this port is not changed. This port is specified by the *CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS* property in *serverindex.xml*.

-nodediscovery

This is an optional parameter. The value *nodediscport* specifies the port on which the node of the node agent discovery service listens.

-nodemulti

This is an optional parameter. The value *nodemultidiscport* specifies the port on which the node agent's multicast discovery service listens.

-verbose

This optional parameter turns on verbose messages, which can be helpful if you need to debug the script.

-help

This optional parameter displays the help message. If you specify this parameter, the script ignores all other parameters.

configureOs400WebServerDefinition command

The **configureOs400WebServerDefinition** command creates a web server definition in a WebSphere Application Server or deployment manager profile.

To run this command, your user profile must have *ALLOBJ authority. For more information about running this command, see the Using command line tools article.

Use an http profile when your web server resides on a different node than the Application Server or deployment manager. This script will generate another script when you run it against an HTTP profile. Use the generated script to create the web server definition in an Application Server or deployment manager profile by running the generated script from the appserver-profile-dir/bin directory, where appserver-profile-dir is the location of your application server or deployment manager. For additional information, see the Selecting a Web server topology diagram and roadmap article.

Syntax

The command syntax is as follows:

```
configureOs400WebServerDefinition
  -webserver.instance.name webserverInstanceName |
  -webserver.name webserverName
  [ -webserver.admin.userID userId ]
  [ -webserver.admin.password password ]
  [ -profileName profile1 ]
  [ -webserver.type webserverType ]
  [ -webserver.host hostName | -webserver.node webserverNode ]
  [ -mapType mapType ]
  [ -webserver.port webserverPort ]
  [ -webserver.admin.port adminPort ]
  [ -was.admin.userId wasUserId ]
  [ -was.admin.password wasPassword ]
  [ -help | -? ]
```

Parameters

The following options are available for the **configureOs400WebServerDefinition** command:

-webserver.instance.name

This parameter is optional when **-webserver.name** is specified. The value *webserverInstanceName* specifies the name of the Web Server instance on the iSeries. The default value is *webserverName*.

-webserver.name

This parameter is optional when **-webserver.instance.name** is specified. The value of *webserverName* specifies the name of the web server definition. The default value is *webserverType_webserverInstanceName*.

-webserver.admin.userId

This is an optional parameter. The value *userId* specifies the user id of the HTTP Administrator on the host name of the Web server.

-webserver.admin.password

This is an optional parameter. The value *password* specifies the password of the HTTP Administrator on the host name of the Web server.

-profileName

This is an optional parameter. The value *profile1* specifies the profile where the web server definition will be configured. The default value is *default*.

-webserver.type

This is an optional parameter. It specifies the type of the Web Server. The value is either IHS or DOMINO. The default value is IHS.

-webserver.host

This is an optional parameter. The `hostName` value specifies the host name of the Web server. The default value is `localhost`.

-webserver.node

This is an optional parameter. The `webserverNode` value specifies the node name of the Web server definition. The default value is `hostName-node`.

-mapType

This is an optional parameter. The value specifies which applications are mapped to the web server definition. You must map applications to a Web server so that the Web server can serve requests to the application. The values allowed are: `MAP_NONE`, `MAP_DEFAULT` or `MAP_ALL`. The default value is `MAP_ALL`. `MAP_DEFAULT` will only map the default application.

-webserver.port

This is an optional parameter. The value specifies the listening port of the Web server. The default is 80.

-webserver.admin.port

This is an optional parameter. The `adminPort` value specifies the admin port of the Web server. The default administrative port for the iseries HTTP Administrative GUI is 2001.

-was.admin.userId

This is an optional parameter. The `wasUserId` value specifies the user ID of the WebSphere Application Server administrator when the server is running in secure mode.

-was.admin.password

This is an optional parameter. The `wasPassword` value specifies the password of the WebSphere Application Server administrator when the server is running in secure mode.

Usage scenario

The following examples demonstrate correct syntax:

Run on an http profile

```
configureOs400WebServerDefinition -profileName profile1.remote -webserver.instance.name WEB1 -webserver.name WEB1
```

The `profile1.remote` profile is an http profile created using the http profile template.

Example output:

```
Web server configuration file: profile_root
/config/WEB1/configureWEB1 was created for remote profile
profile1.remote.
```

You need to copy the "*profile_root*
/config/WEB1/configureWEB1" script to the bin directory of the "Application Server" or "dmgr" profile and run it to create the web server definition "WEB1". Make sure you start the "Application Server" or "dmgr" before you run the script.

The above call created the following script in the `profile1.remote` profile:

```
profile_root/config/WEB1/configureWEB1
```

and indicated the steps needed to complete the web server definition creation.

Run on a non http profile

IHS Example

```
configureOs400WebServerDefinition -profileName profile1 -webserver.instance.name INSTANCE1 -webserver.name SERV1
```

Domino Example

```
configureOs400WebServerDefinition -profileName profile1 -webserver.type DOMINO -webserver.instance.name DOMIN01
```

dspwasinst command

The **dspwasinst** command displays information about a profile and the application servers it contains.

The script displays the following information:

- Application servers. For each application server, the script displays this information:
 - Port numbers
 - Installed applications
 - Status (running or stopped)
 - Job ID (if the application server is running)
- Profile type, location and template used to create.
- Cell name
- Node name
- Network Deployment managing host and port (for a base application server, if you are using Network Deployment)
- Nodes that are managed by the profile (for a Network Deployment profile)

For usage instructions and examples, see the Display profile properties article. To run this script, your user profile must have *ALLOBJ authority.

Syntax

The command syntax is as follows:

```
dspwasinst [-profileName profile] [-server servername] [-help]
```

Parameters

The following options are available for the **dspwasinst** command:

-profileName

This is an optional parameter. The value *profile* specifies the name of the profile that you want to display the properties for. The default value is *default*.

-server

This is an optional parameter. The value *servername* specifies the name of the application server to display. If this value is not specified, all of the application servers in the profile are displayed. To display properties for multiple servers, specify multiple **-server** parameters.

-help

This optional parameter displays the help message. If you specify this parameter, the script ignores all other parameters.

enablejvm command

The **enablejvm** command allows you to configure your application server or a specific profile to use one of the IBM versions of a Java virtual machine (JVM) that is supported on i5/OS.

The following table indicates which JVM versions are supported on which i5/OS releases.

Table 2.

i5/OS Version 5, Release 4, and higher	i5/OS Version 6, Release 1, and higher
i5/OS Java Developer Kit 6.0 JVM, also known as classic JVM, (V5R4(5722JV1) V6R1(5761JV1) option 10)	i5/OS Java Developer Kit 6.0 JVM, also known as classic JVM, (V5R4(5722JV1) V6R1(5761JV1) option 10)
The IBM version of Java Platform, Standard Edition (J2SE) 6.0 32-bit JVM (V5R4(5722JV1) V6R1(5761JV1) option 11)	The IBM version of Java Platform, Standard Edition (J2SE) 6.0 32-bit JVM (V5R4(5722JV1) V6R1(5761JV1) option 11)
	The IBM version of the Java Platform, Standard Edition 6.0 64-bit JVM (5761JV1 option 12)

By default, the product uses the IBM version of the J2SE 6.0 32-bit JVM.

The last i5/OS release on which the i5/OS Java Developer Kit 6.0 JVM will be supported is Version 6, Release 1.

For more information about where to run this command, see the *Using command line tools* topic.

Before you can run the **enablejvm** command, ensure that you meet all the following criteria:

1. The i5/OS system, on which you have the product installed, also has at least two of the supported JVMs installed. If your system has only one JVM installed, there is no reason to run the `enablejvm` command.
2. The Portable Application Solution Environment (PASE) product is installed on your i5/OS system. Both the IBM version of the J2SE 6.0 32-bit JVM and the IBM version of the J2SE 6.0 64-bit JVM require this product. If this product is not already installed, install the PASE product which is option 33 of the i5/OS operating system, and is included with the media for i5/OS Version 5, Release 4, Modification Level 0, and higher.

Syntax

The command syntax is as follows:

```
enablejvm [options]
```

Parameters

The following options are available for the **enablejvm** command:

-jvm *jvm_version*

This is a required parameter. `enablejvm` does not process the request if the target *jvm_version* is not installed. *jvm_version* must be one of the following supported versions:

- `classic`. When this value is entered, the application server is configured to use the i5/OS Java Developer Kit 6.0 JVM.
- `std32`. When this value is entered, the application server is configured to use the IBM version of the J2SE 6.0 32-bit JVM.
- `std64`. When this value is entered, the application server is configured to use the IBM version of the J2SE 6.0 64-bit JVM. This value is valid if you are running the product on i5/OS Version 6, Release 1, or higher.

-profile *profile_name*

This is an optional parameter. If you do not specify the `-profile` parameter, the script enables the product to use the J2SE 5.0 JVM and also updates any existing profiles to use the JVM that is specified for *jvm_version*. The script also updates any existing profiles to use this JVM. New profiles

inherit the JVM settings of the product at the time that they are created. If you do specify the `-profile` parameter, only the profile specified by `profile_name` is updated to use the JVM that is specified for `jvm_version`.

Usage scenario

The following examples demonstrate correct syntax:

```
enablejvm -jvm std32 [-profile profilename]
```

If you did not specify the `-profile` parameter on the **enablejvm** invocation, restart any servers which are running. If you did specify the `-profile` parameter on the **enablejvm** invocation, you must restart any servers that belong to the profile that was changed before the change takes affect.

The `-Xms` or `-Xmx` Java arguments are used to change the initial Java heap size or the maximum Java heap size for an application server. If you previously used the administrative console to add either the `-Xms` or `-Xmx` Java arguments to the Generic JVM arguments setting for an application server, these settings might not be appropriate or even valid for the IBM version of the J2SE 6.0 32-bit JVM. Therefore, after you enable a profile to use the IBM version of the J2SE 6.0 32-bit JVM, the first time that you start an application server under that profile, the application server might fail because either of these Java heap size settings is greater than the 32-bit JVM three gigabyte Java heap size limit. You should review all of your Java heap settings in the administrative console and either change or eliminate them as appropriate.

enbprfwas command

The **enbprfwas** command enables a profile to allow an application server to run under it and optionally changes the group profile to `QEJBSVR`.

It is an alternative to using iSeries Navigator to do the same thing. To run this script, your user profile must have `*ALLOBJ` authority.

Syntax

The command syntax is as follows:

```
enbprfwas -profile <user profile> [-chggrprrf]
```

Parameters

The following options are available for the **enbprfwas** command:

-profile

This is a required parameter. The value `<user profile>` specifies the name of the profile that you want to enable to run application servers.

-chggrprrf

This is an optional parameter. If you specify this parameter, the command changes the group profile of `<user profile>` to `QEJBSVR`.

heapMonitor command

The **heapMonitor** command manages the iSeries WebSphere Custom Service Heap Monitor, which sends messages to `QSYSOPR` when the WebSphere Java heap is close to exceeding or exceeds the storage pool size.

Product

This script is available in WebSphere Application Server, WebSphere Application Server - Express, and WebSphere Application Server, Network Deployment.

Authority

To run this script, your user profile must have *ALLOBJ authority.

Syntax

```
heapMonitor [-profileName profile] [wsadmin_options]
[-server.name server_name] [-enable | -disable | -status | -viewlog] [-help]
```

- **-profileName**
The name of the profile to manage. The default profile is used if no profile is specified.
- **-server.name**
The name of the application server to manage. The default value is the name of the profile, or "server1" if the profile is named "default."
- **-enable**
Enables heap monitoring on a server. If heap monitoring is already enabled, it does nothing.
- **-disable**
Disables heap monitoring on a server. If heap monitoring is already disabled, it does nothing.
- **-status**
Displays whether heap monitoring is enabled on a server.
- **-viewlog**
Displays heap monitor messages since the last startup of the application server.
- **-help**
Displays this help message.

ivt command

The install verification (ivt) script verifies that the application server for an instance is functioning correctly.

For a WebSphere Application Server instance, the ivt script verifies that a JavaServer page, a servlet and an enterprise bean can be successfully invoked. For a WebSphere Application Server Network Deployment instance, the ivt script verifies that the deployment manager can be contacted successfully. If the application server is not running, the ivt script attempts to start it before verification begins. For WebSphere Application Server instances, the ivtApp application must be installed in the server. By default, this application is installed in the default server that is created when you create an instance. To run this script, your user profile must have *ALLOBJ authority.

Syntax

The command syntax is as follows:

```
ivt [-instance instance]
```

Parameters

The following options are available for the **ivt** command:

- **-instance**
This is an optional parameter. The value instance specifies the name of the instance for which you want to verify the installation. The default value is default (or the instance specified by the WAS_OS400_INSTANCE variable). For information about the WAS_OS400_INSTANCE variable, see the Qshell environment variables article.

Usage scenario

The following example verifies the default instance:

```
ivt
```

The following example verifies the devinst instance:

```
ivt -instance devinst
```

port validator tool

The port validator tool verifies your WebSphere Application Server configuration to ensure that you do not have port conflicts between WebSphere Application Server profiles and products.

The port validator tool is one of the tools available with the servicetools script.

Syntax

```
servicetools [ parameters ] -portconflict [ -products product_list ]  
  [ -profiles profile_list ] [ -ports port_list ]  
  [ -comparetoproduct compare_prod ] [ -comparetoprofile compare_inst ]  
  [ -xmloutput output_file ]
```

Parameters

parameters

This optional parameter list specifies parameters for the servicetools script.

The servicetools parameters must be specified before the -portconflict parameter.

-portconflict

Specify this parameter to invoke the port validator tool.

-products

This is an optional parameter. The value *product_list* specifies the product or products for which you want to check for port conflicts. If you specify multiple products, separate them with a colon (:). Valid products are WAS61Base, WAS70ND, WAS70Express, WAS70Base, WAS61ND, WAS61Express, WAS60Base, WAS60ND, WAS60Express, WAS51Base, WAS51ND, WAS51Express, WAS51PME, WAS51PMEND, WAS50Base, WAS50ND, WAS50Express, WAS40Adv, WAS40AEs, and ALL.

The port validator tool uses the -products and -profiles parameters to determine the set of ports that it checks for conflicts. The tool generates a default set of ports that includes the ports used by the specified products and profiles. If you do not specify this parameter, the port validator tool checks for port conflicts in the products specified by the servicetools -products parameter.

If you specify this parameter and the -products parameter for servicetools, the port validator tool checks only the products that you specify for the -portconflict -products parameter. You must specify the -product parameter for either the servicetools script or the port validator tool, or both.

-profiles

This is an optional parameter. The value *profile_list* specifies the profile or profiles for which you want to check for port conflicts. If you specify multiple profiles, separate them with a colon (:).

The port validator tool uses the -products and -profiles parameters to determine the default set of ports that it checks for conflicts. The tool generates a default set of ports that includes the ports used by the specified products and profiles. If you do not specify this parameter, the port validator tool checks for port conflicts in the profiles specified by the servicetools -profiles parameter, profiles within the products specified by the -products parameter.

If you specify this parameter and the -profiles parameter for servicetools, the port validator tool checks only the profiles that you specify for the -portconflict -profiles parameter. If you do not specify the -profiles parameter for either the service tools script or the port validator tool, the tool checks for port conflicts in all profiles in the specified products.

-ports

This is an optional parameter. The value *port_list* is a list of ports that you want to check for conflicts. If you specify this parameter, the tool generates a set of ports based on the value *port_list*. The tool checks for conflicts between this set of ports and the default set that is generated based on the `-products` and `-profiles` parameters.

You can specify individual ports, or port blocks. Separate ports or port blocks with a colon (:). For example, to check for conflicts on ports 6680, 7600-7610, and 13320, include specify this parameter: `-ports 6680:7600-7610:13320` If you do not specify this parameter, the port validator tool uses the values specified by the `-products`, `-profiles`, `-comparetoproduct`, and `-comparetoprofile` parameters to check for conflicts. You cannot specify the `-ports` parameter with the `-comparetoproduct` parameter or with the `-comparetoprofile` parameter.

-comparetoproduct

This is an optional parameter. The port validator tool compares port usage between the product specified by *compare_prod* and the products and profiles specified by `-products` and `-profiles`. If you specify this parameter, the port validator tool generates a set of ports based on the specified product. The tool checks for conflicts between this set of ports and the default set that is generated based on the `-products` and `-profiles` parameters.

You cannot specify the `-comparetoproduct` parameter with the `-ports` parameter.

-comparetoprofile

This is an optional parameter. The port validator tool compares port usage between the profile specified by *compare_inst* and the products and profiles specified by `-products` and `-profiles`. If you specify this parameter, the port validator tool generates a set of ports based on the specified profile. The tool checks for conflicts between this set of ports and the default set that is generated based on the `-products` and `-profiles` parameters.

You cannot specify the `-comparetoprofile` parameter with the `-ports` parameter.

-xmloutput

This is an optional parameter. The value *output_file* is the fully qualified path of an XML file. If you specify this parameter, the port validator tool writes the output in XML format to the specified file, in addition to writing the output to the display.

prerequisite validator tool

The prerequisite validator tool verifies your WebSphere Application Server installation, ensures that you have the appropriate prerequisite software and PTFs installed, and verifies certain configuration settings required by the product.

The prerequisite validator tool can also be run from the `checkprereqs` script. See [The checkprereqs script](#) for more information.

For descriptions of the syntax and parameters that apply to the `servicetools` script, see [The servicetools script](#).

Syntax

The syntax to run the prerequisite validator tool is:

```
servicetools [ parameters ] -checkprereqs [ -xmlinput xml_input_file ]  
[ -xmloutput xml_output_file ] [ -verbose ]
```

Parameters

The parameters for the prerequisite validator tool are:

parameters

This optional parameter list specifies parameters for the servicetools script. See The servicetools script for more information. The servicetools parameters must be specified before the -checkprereqs parameter.

-checkprereqs

This parameter invokes the prerequisite validator.

-xmlinput

This is an optional parameter. The value xml_input_file specifies the fully qualified path of the XML file that lists the prerequisites that the script checks. If this parameter is not specified, the script uses the default XML input file that is included in the ServiceTools.jar file.

-xmloutput

This is an optional parameter. The value xml_output_file specifies the fully qualified path of the XML file to which the script writes the results. If neither this parameter nor the -output parameter is specified, output information is written to the display.

-verbose

This is an optional parameter. If you specify this parameter, the script displays all result information. If you do not specify this parameter, the script does not display successful result information.

removeOs400WebServerDefinition command

The **removeOs400WebServerDefinition** command deletes a web server definition from the configuration repository of a Standalone or Deployment Manager profile.

To run this command, your user profile must have *ALLOBJ authority. For more information about running this command, see the Using command line tools article.

Syntax

The command syntax is as follows:

```
removeOs400WebServerDefinition
  -webserver.name webserverName
  -webserver.node webserverNode
  [-profileName profile1]
  [-removeNode]
  [-help | -?]
```

Parameters

The following options are available for the **removeOs400WebServerDefinition** command:

-webserver.name

This is a required parameter. The value webserverName specifies the web server definition name you want to delete.

-profileName

This is an optional parameter. The value profile1 specifies the profile whose web server definition is to be deleted. The default value is default.

-webserver.node

This is a required parameter. The value webserverNode specifies the node under which the web server is defined.

-removeNode

This is an optional parameter. Indicates to remove the Web server node. If you do not specify the -removeNode parameter, the Web server node is not removed.

Usage scenario

The following example deletes the web server definition, WEBSERVER1, under the unmanaged node, MYHOST-node, and removes the node. It also updates the deployment data for any application that was mapped to the above Web server definition.

```
remove0s400WebServerDefinition -profileName profile1 -webserver.name WEBSERVER1  
-webserver.node MYHOST-node -removeNode
```

rvkwasaut command

The **rvkwasaut** command revokes user authority to a profile and the objects associated with it.

For usage instructions and examples, see the Revoke authority to a profile article. To run this script, your user profile must have *ALLOBJ authority.

Syntax

The command syntax is as follows:

```
rvkwasaut [-profileName profile] [-user usrprf | -authlist none]  
[-object path] [-recursive] [-verbose] [-help]
```

Parameters

The following options are available for the **rvkwasaut** command:

-profileName

This is an optional parameter. The value *profile* specifies the name of the profile to which you are revoking authority. The default value is *default*.

-user

The value *usrprf* specifies the user profile from which you are revoking authority. To revoke authority from multiple user profiles, specify all of the user profiles with a single *-user* parameter. Enclose the list of profiles in double quotation marks (""). For example, to revoke authority from *usrprf1* and *usrprf2*, specify *-user "usrprf1 usrprf2"*. You must specify *-user*, *-authlist*, or both.

-authlist

The value *none* specifies that the current authorization list associated with the object should be removed. If you use the *-authlist* parameter, you must specify the value *none*. You must specify either *-user*, *-authlist*, or both.

-object

This is an optional parameter. The value *path* specifies the subdirectory or partially qualified object name to which you are revoking authority. The profile root is prepended to the value to get the fully-qualified path. If you do not specify this parameter, the default value is the profile root. To revoke authority to multiple objects, you must run the script for each object.

-recursive

This optional parameter specifies whether to revoke authority to all subdirectories. If you do not specify this parameter, authority is revoked only to the object specified with the *-object* parameter, or the profile root directory if the *-object* parameter is not specified. This parameter applies to all objects specified with *-object* parameters.

-verbose

This optional parameter turns on verbose messages, which can be helpful if you need to debug the script.

-help

This optional parameter displays the help message. If you specify this parameter, the script ignores all other parameters.

servicetools command

The **servicetools** command includes serviceability tools that you can use to troubleshoot WebSphere Application Server.

To run this script, your user profile must have *ALLOBJ authority.

Syntax

The command syntax is as follows:

```
servicetools [-products product_list [-profiles profile_list]]  
  [-input input_file] [-output output_file] [-trace trace_file]  
  [-tools tool_options] [-help]
```

Parameters

The following options are available for the **servicetools** command:

-products

This is an optional parameter. The value *product_list* specifies the list of products for which you want to run the tools. If you specify multiple products, separate them with a colon (:). Valid products are WAS61Base, WAS61ND, WAS61Express, WAS60Base, WAS60ND, WAS60Express, WAS51Base, WAS51ND, WAS51Express, WAS51PME, WAS51PMEND, WAS50Base, WAS50ND, WAS50Express, WAS40Adv, WAS40AEs, and ALL. If you do not specify this parameter, you must specify the **-products** parameter for each tool as appropriate.

Not all tools require that you specify a product. This parameter applies only to those tools that require that you specify a product.

-profiles

This is an optional parameter. The value *profile_list* specifies the profile or profiles for which you want to run the tools. If you specify multiple profiles, separate them with a colon (:). If you do not specify this parameter, the tools run for all of the profiles in the products specified by the **-products** parameter.

-input

This is an optional parameter. The value *input_file* specifies an alternate XML Configuration file for the service tools. By default, the **servicetools** script uses the *validator.xml* file that is included in the *ServiceTools.jar* file.

-output

This is an optional parameter. The value *output_file* specifies a file to which the script writes standard output. If this value is not specified, the script writes the standard output to the display.

-trace

This is an optional parameter. The value *trace_file* specifies a file to which the script writes trace information. If you do not specify this parameter, the script does not generate trace information.

-tools

This parameter specifies the tool or tools that you want to run. The value *tools* specifies the tool that you want to run. The value *tool_options* specifies the parameters for the tool. These are the valid values for **-tools**:

- **-checkprereqs** If you specify this parameter, the script starts the prerequisite validator tool. For information about this tool and its parameters, see *The prerequisite validator tool*.
- **-portconflict** If you specify this parameter, the script starts the port validator tool. For information about this tool and its parameters, see *The port validator tool*.

You must specify all other parameters before you specify the **tools** parameter. You must specify the options for each tool after its parameter. You can specify more than one tool when you run the **servicetools** script. To run multiple tools, use this format:

```
servicetools parameters -tool1 tool1_options -tool2 tool2_options
```

-help

This optional parameter displays the usage statement for the script.

updwashost command

The updwashost script updates a profile if the host name of the server is changed.

To run this script, your user profile must have *ALLOBJ authority.

Syntax

The command syntax is as follows:

```
updwashost -profileName profile [-newhost new_host]  
[-changenode -oldhost old_host] [-regenplugin] [-verbose] [-help]
```

Parameters

The following options are available for the **updwashost** command:

-profileName

This is a required parameter. The value *profile* specifies the name of the profile that you want to update. To update the host name for all profiles, specify ALL.

-newhost

This is an optional parameter. The value *new_host* specifies the new host name to use for the profile. The default value is the current host name of the server.

-changenode

This is an optional parameter. If you specify this parameter, the script changes the node name in the profile. If you specify this parameter, the -oldhost parameter is required.

If the node of the profile node is federated into a deployment manager, it is recommended that you not include this parameter, as it might break references to the node name.

-oldhost

This parameter is required if the -changenode parameter is specified, and not valid otherwise. The value specifies the old host name to use when changing the node name for the profile. The value specified will be replaced in the node name with the value specified in the -newhost parameter.

-regenplugin

This optional parameter will call the **GenPluginCfg** command after updating the host for the profile to regenerate the HTTP server plug-in configuration.

-verbose

This optional parameter turns on verbose messages, which can be helpful if you need to debug the script.

-help

This optional parameter displays the help message. If you specify this parameter, the script ignores all other parameters.

grtwasaut command

The grtwasaut command grants a user authority to a profile and the objects associated with it.

For usage instructions and examples, see the Grant authority to a profile article. To run this script, your user profile must have *ALLOBJ authority.

Syntax

The command syntax is as follows:

```
grtwasaut [-profileName profile] {-user usrprf | -authlist authlist}
{-dtaaut dataAuth | -objaut objectAuth} [-object path]
[-recursive] [-verbose] [-help]
```

When you run the **grtwasaut** command, you must specify the following parameters:

- -user, -authlist, or both
- -dtaaut, objaut, or both

Parameters

The following options are available for the **grtwasaut** command:

-profileName

This is an optional parameter. The value *profile* specifies the name of the profile to which you are granting authority. The default value is *default*.

-user

The value *usrprf* specifies the user profile to which you are granting authority. To grant authority to multiple user profiles, specify all of the user profiles with a single -user parameter. Enclose the list of profiles in double quotation marks ("). For example, to grant authority to usrprf1 and usrprf2, specify -user "usrprf1 usrprf2". You must specify -user, -authlist, or both.

-authlist

The value *authlist* specifies the authorization list to which you are granting authority. You must specify -user, -authlist, or both.

-dtaaut

The value *dataAuth* specifies the data authorities that you are granting to the user specified by the -user parameter. Valid values are none, rwx, rx, rw, wx, r, w, x, exclude, autl, and same. The value that you specify replaces the user's current data authorities to the object. You must specify -dtaaut, -objaut, or both. For more information on the values for this parameter, see the CHGAUT (Change Authority) command description.

-objaut

The value *objAuth* specifies the object authorities that you are granting to the user specified in the -user parameter. Valid values are none, all, objexist, objmgt, objalter, objref, and same. The value that you specify replaces the user's current object authorities to the object. You must specify -dtaaut, -objaut, or both. For more information on the values for this parameter, see the CHGAUT (Change Authority) command description.

-object

This is an optional parameter. The value *path* specifies the subdirectory or partially qualified object name to which you are granting authority. The profile root is prepended to the value to get the fully-qualified path. If you do not specify this parameter, the default value is the profile root. To grant authority to multiple objects, you must run the script for each object.

-recursive

This optional parameter specifies whether to grant authority to all subdirectories. If you do not specify this parameter, authority is granted only to the object specified with the -object parameter, or the profile root directory if the -object parameter is not specified. This parameter applies to all objects specified with -object parameters.

-verbose

This optional parameter turns on verbose messages, which can be helpful if you need to debug the script.

-help

This optional parameter displays the help message. If you specify this parameter, the script ignores all other parameters.

registerNode command

Use the registerNode command to register a standalone node with an administrative agent so that the administrative agent can manage the node.

Run the registerNode command from the bin directory of the administrative agent server to register a node with the administrative agent. When you run the command, the standalone node is converted into a node that the administrative agent manages. The administrative agent and the node being registered must be on the same system. You can only run the command on an unfederated node. If the command is run on a federated node, the command exits with an error.

If the administrative console or the management Enterprise JavaBeans (EJB) applications of the application server being registered are enabled, the node registration process disables them.

Syntax

The registerNode command syntax is as follows:

```
registerNode [options]
```

Parameters

The following options are available for the registerNode command:

-conntype<JSR160RMI|PCIRMI|SOAP>

The optional connector type used to connect to the administrative agent to initiate node registration. The default is SOAP.

Note: You should eventually switch from the RMI connector to the JSR160RMI connector because support for the RMI connector is deprecated.

-host *host_name*

An optional parameter that specifies the host name of the administrative agent.

-name *managed_node_name*

An optional parameter that specifies the name of the managed node after the node is registered with the administrative agent.

-nodepassword *node_password*

An optional parameter that specifies the password of the node that you are registering. Specify this parameter if security is on at the node and the password is different than the administrative agent password. Use this parameter with the -nodeusername parameter. The -nodeusername and -nodepassword parameters are used to stop all servers on the node.

-nodeusername *node_user_name*

An optional parameter that specifies the use name of the node that you are registering. Specify this parameter if security is on at the node and the user name is different than the administrative agent user name. Use this parameter with the -nodepassword parameter. The -nodeusername and -nodepassword parameters are used to stop all servers on the node.

-openConnectors *connectors*

An optional parameter that specifies a list of connectors separated by commas of connectors that the administrative agent will open. By default, all connectors are opened.

-port *port_number*

An optional parameter that specifies the port number of the administrative agent connector port.

The default port number is 8878 for the default SOAP port of the administrative agent. SOAP is the default Java Management Extensions (JMX) connector type for the command. If you have multiple product installations or multiple profiles, the SOAP port might be different than 8878. Examine the administrative agent SystemOut.log file to see the current ports in use.

-profilePath *profile_path*

A required parameter that specifies the path to the profile of the application server node to be registered.

-trace

An optional parameter that provides tracing output for the registerNode command.

-portsFile *ports_file*

An optional parameter that specifies the path to a file that defines port settings for the newly registered node.

During node registration, the registerNode command uses an automatically generated set of recommended ports if you do not specify the -portsFile parameter. The recommended port values can be different than the default port values based on the availability of the default ports.

The format of the *ports_file* file that you specify is the same as the portdef.props file, except that only the following ports are used: SOAP_CONNECTOR_ADDRESS, RMI_CONNECTOR_ADDRESS, JSR160RMI_CONNECTOR_ADDRESS, and IPC_CONNECTOR_ADDRESS.

-profileName *profile_name*

An optional parameter that specifies the profile name of the administrative agent.

-username *username*

An optional parameter that specifies the user ID to log onto the administrative agent.

-password *password*

An optional parameter that specifies the password to log onto the administrative agent.

-help

An optional parameter that prints a usage statement.

-? An optional parameter that prints a usage statement.

Usage scenarios

The following examples demonstrate correct syntax. Commands are split on multiple lines for printing purposes.

```
registerNode -conntype SOAP -port 8878  
-profilePath user_data_root/profiles/AppSrv01
```

deregisterNode command

Use the deregisterNode command to unregister a node from an administrative agent so that you can use the node standalone or register the node with another administrative agent.

Run the deregisterNode command from the bin directory of the administrative agent. The node must have been previously registered with the administrative agent. When you unregister a node, the node configuration is retained, but is marked as not registered with the administrative agent.

If the node that you unregister had the administrative console or management Enterprise JavaBeans (EJB) applications installed prior to registering the node, they are re-enabled.

Syntax

The deregisterNode command syntax is as follows:

```
deregisterNode [options]
```

Parameters

The following options are available for the deregisterNode command:

-conntype<JSR160RMI|PCIRMI|SOAP>

The optional connector type used to connect to the administrative agent to initiate node deregistration. The default is SOAP.

Note: You should eventually switch from the RMI connector to the JSR160RMI connector because support for the RMI connector is deprecated.

-host *host_name*

An optional parameter that specifies the host name of the administrative agent.

-nodepassword *node_password*

An optional parameter that specifies the password of the node that you are unregistering. Specify this parameter if security is on at the node and the password is different than the administrative agent password. Use this parameter with the `-nodeusername` parameter. The `-nodeusername` and `-nodepassword` parameters are used to stop all servers on the node.

-nodeusername *node_user_name*

An optional parameter that specifies the use name of the node that you are unregistering. Specify this parameter if security is on at the node and the user name is different than the administrative agent user name. Use this parameter with the `-nodepassword` parameter. The `-nodeusername` and `-nodepassword` parameters are used to stop all servers on the node.

-port *port_number*

An optional parameter that specifies the port number of the administrative agent connector port.

The default port number is 8878 for the default SOAP port of the administrative agent. SOAP is the default Java Management Extensions (JMX) connector type for the command. If you have multiple product installations or multiple profiles, the SOAP port might be different than 8878. Examine the administrative agent SystemOut.log file to see the current ports in use.

-profilePath *profile_path*

A required parameter that specifies the path to the profile of the base node to be unregistered.

-trace

An optional parameter that provides tracing output for the `deregisterNode` command.

-profileName *profile_name*

An optional parameter that specifies the profile name of the administrative agent.

-username *username*

An optional parameter that specifies the user ID of the node to be unregistered.

-password *password*

An optional parameter that specifies the password of the node to be unregistered.

-help

An optional parameter that prints a usage statement.

-? An optional parameter that prints a usage statement.

Usage scenarios

The following examples demonstrate correct syntax. Commands are split on multiple lines for printing purposes.

```
deregisterNode -conntype SOAP -port 8878  
-profilePath user_data_root/profiles/AppSrv01
```


revokeCertificate command

The revokeCertificate command uses an implementation class that is passed to communicate with a certificate authority (CA) server to revoke a certificate. Processing this command sends a revocation request to the CA server to mark this certificate as revoked.

Location

Issue the command from the *profile_root/bin* directory.

Syntax

The command syntax is as follows:

(The command is split on multiple lines for printing purposes.)

```
revokeCertificate -host<caHost> -port<caPort> -username<caUserName> -password<caPassword>  
-revocationPassword<revocationPassword> -keystoreAlias<keystoreAlias> -alias<certificateAlias>  
-pkImplClass<customCAClient>[options]
```

Required Parameters

The following required parameter are used with the revokeCertificate command:

-host *caHost*

Specifies the target certificate authority host to which the request is sent.

-port *caPort*

Specifies the target port to connect to.

-username *caUserName*

Specifies the user name used to gain access to the certificate authority.

-password *caPassword*

Specifies the password used to authenticate with the certificate authority.

-revocationPassword *revocationPassword*

Specifies the password that is to be set on the certificate returned by the certificate authority. The revocation password is sent to the certificate authority during each request and is associated with each certificate that is issued. To later revoke a certificate, the same revocation password must be sent during a revokeCertificate request.

keyStoreAlias *keyStoreAlias*

Specifies the name of the keystore that is located in the *ssl.client.props* file for the profile to which the CA signed certificate is added. This file is usually the *ClientDefaultKeyStore* file for either a managed or unmanaged environment.

-alias *certificateAlias*

Specifies The alias of the certificate request to be revoked. The certificate is stored in the keystore specified on the request.

-pkImplClass *custom CA Client*

A class that implements the *WSPKIClient* interface. The implementation class handles all the communication to the CA server. This can be a custom class or a class provided with the product.

Optional Parameters

The following options are available for the revokeCertificate command:

-revocationReasonUsage *revocation reason*

The reason for revoking the certificate. The default value is "unspecified".

-customAttrs *customAttr1=value;customAttr2=value;...*

A semi-colon separated list of custom name=value pairs to be passed in to the custom implementation class. This parameter provides a way to pass custom information to the implementation class. The 'attr' and 'value' pairs are converted to a hash map and passed to the implementation class.

-logfile *filename*

Overrides the default trace file. By default, the trace appears in the profiles/profile_name/log/caClient.log file.

-trace

When specified, **-trace** enables tracing of the trace specification necessary to debug this component. By default, the trace appears in the profiles/profile_name/log/caClient.log file.

-replaceLog

An option to cause the existing trace file to be replaced when the command is executed. -quit

-quiet

An option to suppress most messages from printing out on the console.

-help

The option to print a usage statement

-? The option to print a usage statement

Usage

The following example performs a revokeCertificate:

```
revokeCertificate -host localhost -port 1077
-username pkuser -password webspherepki -alias cert1 -keyStoreAlias ClientDefaul
ltKeyStore -revocationPassword webspherepki
CWPKI0403I: Trace is being logged to the following location:
      C:\opt\WebSphere\AppClient\logs\caClient.log
CWPKI0461I: Revoking a CA signed certificate.
CWPKI0462I: CA Signed Certificate Revoked [Issued By: O=IBM, C=US, Issued To:
      CN=mycn, O=ibm, C=us, Not Before: Thu Feb 22 09:07:53 CST 2007, Not
      After: Sat Feb 16 10:09:19 CST 2008] for reason: unspecified
```

requestCertificate command

The requestCertificate command uses an implementation class that is passed in to communicate with a certificate authority (CA) server to request a CA signed certificate. The command then adds the certificate to a supplied keystore.

The requestCertificate command can use a predefined certificate request that was created with the createCertRequest command or it creates the certificate request itself. Depending on the CA server that the command is targeted for, a completed signed request can be returned; or the CA server could accept the request and require that a call be made at a later time to get the certificate with the queryCertificate command.

Location

Issue the command from the *profile_root/bin* directory.

Syntax

The command syntax is as follows:

(The following command is split on multiple lines for printing purposes.)

```
requestCertificate -host<caHost> -port<caPort> -username<caUserName> -password<caPassword>
-revocationPassword<revocationPassword> -keystoreAlias<keystoreAlias>
-pkiImplClass<customCAClient>[options]
```

Required Parameters

The following required parameter are used with the requestCertificate command:

-host *caHost*

Specifies the target certificate authority host to which the request will be sent.

-port *caPort*

Specifies the target port on which to connect.

-username *caUserName*

The user name used to gain access to the certificate authority.

-password *caPassword*

The password used to authenticate with the certificate authority.

-revocationPassword *revocationPassword*

The password that is to be set on the certificate returned by the certificate authority. The revocation password is sent to the certificate authority during each request and is associated with each certificate that is issued. To later revoke a certificate, the same revocation password must be sent during a revokeCertificate request.

keyStoreAlias *keyStoreAlias*

The name of the keystore that is located in the ssl.client.props file for the profile to which the CA signed certificate is added. This will typically be the ClientDefaultKeyStore file for either a managed or unmanaged environment.

-pkimplClass *custom CA client*

A class that implements the WSPKIClient interface. The implementation class handles all the communication to the CA server. This could be a custom class or a class provided with the product.

Optional Parameters

The following options are available for the requestCertificate command:

-certReqPath *certificate request file*

A path to an existing PKCS10 certificate request saved in a BASE64 encoded file. If no request is specified a PKCS10 certificate request will be created automatically. In that case it is required to specify a "subjectDN" and "alias" option. By default the request will be created in the same location as the keyStore specified in the request. This will typically be in the /profile_name/etc/ directory for either a managed or unmanaged environment.

-subjectDN *subjectDN*

The distinguished name to be used for the PKCS10 certificate request. The distinguished name must contain the CN field. This option is only required if you do not specify the –certReqPath option, or if the –certReqPath option points to a file that does not exist.

-alias *certificateAlias*

The alias used to store the PKCS10 certificate request certificate in the keyStore specified on the request. Note that the CA signed certificate is stored under the same alias and will replace the cert request certificate when received. This option is only required if you do not specify the –certReqPath option, or if the –certReqPath option points to a file that does not exist.

-keySize *key size*

The size of the key. This option is only used valid if creating a PKCS10 certificate request in-band. Default size is 1024. Valid values include 512, 1024, and 2048

-keyUsage

A semi-colon separated list of extended key usage strings. This option is only valid if creating a PKCS10 certificate request in-band.

-extKeyUsage *extKeyUse1;extKeyUse2;...*

A semi-colon separated list of extended key usage strings. This option is only valid if creating a PKCS10 certificate request in-band.

-customAttrs *customAttr1=value;customAttr2=value;...*

A semi-colon separated list of custom name=value pairs to be passed in to the custom implementation class. This provides a way to pass custom information to the implementation class. The 'attr' and 'value' pairs will be converted to a hash map and passed along to the implementation class.

-retryInterval *retry interval*

The time period in seconds between retries of queries to the CA for a CA signed certificate.

-retryLimit *retry limit*

The total number of times to retry a query request to the CA.

-logfile *filename*

Overrides the default trace file. By default, the trace appears in the profiles/profile_name/log/caClient.log. file.

-trace

When specified, this enables tracing of the trace specification necessary to debug this component. By default, the trace will appear in the profiles/profile_name/log/caClient.log file.

-replaceLog

Causes the existing trace file to be replaced when the command is executed. -quit

-quiet

Suppresses most messages from printing out on the console.

-help

Prints a usage statement

-? Prints a usage statement

Usage

The following example performs a requestCertificate:

```
requestCertificate -host localhost -port 1077
  -username pkiuser -password webspherepki -revocationPassword webspherepki -keyS
toreAlias ClientDefaultKeyStore -certReqPath C:\opt\WebS
phere\AppClient\etc\certReq26924.req -trace
CWPKI0403I: Trace is being logged to the following location:
  C:\opt\WebSphere\AppClient\logs\caClient.log
CWPKI0455I: Requesting a CA signed certificate.
CWPKI0456I: CA Signed Certificate Received [Issued By: O=IBM, C=US, Issued To:
  CN=mycn, O=ibm, C=us, Not Before: Thu Feb 22 09:07:53 CST 2007, Not
  After: Sat Feb 16 10:09:19 CST 2008]
```

createCertRequest command

The createCertRequest command creates a PKCS10 certificate request and stores it in a client keystore so that it can be used to send to a certificate authority (CA) server using the requestCertificate command line utility.

Location

Issue the command from the *profile_root/bin* directory.

Syntax

The command syntax is as follows:

```
createCertRequest -keyStoreAlias<keystoreAlias> -subjectDN<subjectDN> -alias<certificateAlias> [options]
```

Required Parameters

The following required parameter are used with the createCertRequest command:

-keyStoreAlias *keyStoreAlias*

Specifies the name of the keystore that is located in the ssl.client.props file for the profile to which the CA signed certificate is added. This is the name of the ClientDefaultKeyStore file for either a managed or unmanaged environment.

-subjectDN *subjectDN*

Specifies the distinguished name (DN) to be used for the PKCS10 certificate request. The DN must contain the CN, O and C fields at a minimum.

-alias *certificateAlias*

Specifies the alias used to store the PKCS10 certificate request certificate in the keystore specified on the request.

Note: the CA signed certificate is stored under the same alias and replaces the cert request certificate when received.

Optional Parameters

The following options are available for the createCertRequest command:

-keySize *key size*

An option that specifies the size of the key. This option is only used valid if creating a PKCS10 certificate request in-band. Default size is 1024. Valid values include 512, 1024, and 2048

-certValidity *valid days*

The time period of certificate validity. Time period is measured from current date. This option is only valid if creating a PKCS10 certificate request in-band. Default value is 365 days.

-subjectAltNames *altName1;altName2;...*

A semi-colon separated list of subject alternate names. This option is only used if creating a PKCS10 certificate request in-band.

-keyUsage *keyUse1;keyUse2;...*

A semi-colon separated list of key usage strings. This option is only valid if creating a PKCS10 certificate request in-band.

-extKeyUsage *extKeyUse1;extKeyUse2;...*

A semi-colon separated list of extended key usage strings. This option is only valid if creating a PKCS10 certificate request in-band.

-logfile *filename*

The logfile that overrides the default trace file. By default, the trace appears in the profiles/profile_name/log/caClient.log. file.

-trace

When specified, **-trace** enables tracing of the trace specification necessary to debug this component. By default, the trace will appear in the profiles/profile_name/log/caClient.log file.

-replaceLog

An option to cause the existing trace file to be replaced when the command is executed.

-quiet

An option to suppress most messages from printing out on the console.

-help

The option to print a usage statement

-? The option to print a usage statement

Usage

The following example creates a PKCS10 certificate request for a client that can be used to send to a CA :

```
createCertRequest -keyStoreAlias ClientDefaultKeyStore -subjectDN CN=mycn,o=ibm,c=us -alias cert1
CWPKI0403I: Trace is being logged to the following location:
           C:\opt\WebSphere\AppClient\logs\caClient.log
CWPKI0422I: Generating a PKCS10 certificate request
CWPKI0421I: A PKCS10 certificate was successfully created. The request
           is stored in file:
           C:\opt\WebSphere\AppClient\etc\certReq26924.req
```

queryCertificate command

The queryCertificate command uses an implementation class that is passed to communicate with a certificate authority (CA) server and query a certificate.

The queryCertificate command checks to see if the certificate is complete. If the certificate is complete, then the CA certificate is stored in the client keystore. If the certificate is not complete, the certificate request remains in the keystore and the queryCertificate command can be called at some later time to determine if the certificate is complete.

Location

Issue the command from the *profile_root/bin* directory.

Syntax

The command syntax is as follows:

(The command is split on multiple lines for printing purposes.)

```
queryCertificate -host<caHost> -port<caPort> -username<caUserName> -password<caPassword>
-alias<certificateAlias> -keystoreAlias<keystoreAlias>
-pkiImplClass<customCAClient>[options]
```

Required Parameters

The following required parameter are used with the queryCertificate command:

-host *caHost*

Specifies the target certificate authority host to which the request is sent.

-port *caPort*

Specifies the target port to connect to.

-username *caUserName*

Specifies the user name used to gain access to the certificate authority.

-password *caPassword*

Specifies the password used to authenticate with the certificate authority.

-alias *certificateAlias*

Specifies The alias of the certificate to be queried.

keyStoreAlias*keyStoreAlias*

Specifies the name of the keystore that is located in the ssl.client.props file for the profile to which the CA signed certificate is added. This name is the ClientDefaultKeyStore file for either a managed or unmanaged environment.

-pkiImplClass *custom CA client*

A class that implements the WSPKIClient interface. The implementation class handles all the communication to the CA server. This can be a custom class or a class provided with the product.

Optional Parameters

The following options are available for the queryCertificate command:

-customAttrs *customAttr1=value;customAttr2=value;...*

A semi-colon separated list of custom name=value pairs to be passed in to the custom implementation class. This parameter provides a way to pass custom information to the implementation class. The 'attr' and 'value' pairs are converted to a hash map and passed along to the implementation class.

-retryInterval *retry interval*

The time period in seconds between retries of queries to the CA server for a CA signed certificate.

-retryLimit *retry limit*

The total number of times to retry a query request to the CA server.

-logfile *filename*

The logfile that overrides the default trace file. By default, the trace appears in the profiles/profile_name/log/caClient.log file.

-trace

When specified, **-trace** enables tracing of the trace specification necessary to debug this component. By default, the trace appears in the profiles/profile_name/log/caClient.log file.

-replaceLog

An option to cause the existing trace file to be replaced when the command is executed.

-quiet

An option to suppress most messages from printing out on the console.

-help

The option to print a usage statement

Usage

The following example performs a queryCertificate:

```
queryCertificate -host localhost -port 1077 -
username pkiuser -password webspherepki -alias C:\opt\WebSphere\AppClient\
etc\certReq26924.req -keyStoreAlias ClientDefaultKeyStore
CWPKI0403I: Trace is being logged to the following location:
          C:\opt\WebSphere\AppClient\logs\caClient.log
CWPKI0418E: The following error occurred while querying the CA for a signed
certificate: CWPKI0463I: Action "query" not supported by this
implementation.
```

Chapter 3. Using Ant to automate tasks

To support using Apache Ant with Java Platform, Enterprise Edition (Java EE) applications running on the application server, the product provides a copy of the Ant tool and a set of Ant tasks that extend the capabilities of Ant to include product-specific functions. Ant has become a very popular tool among Java programmers.

About this task

Apache Ant is a Java-based build tool. In theory, it is similar to Make, but Ant is different. Instead of a model in which it is extended with shell-based commands, Ant is extended using Java classes. Instead of writing shell commands, XML-based configuration files are used. These files reference a target tree in which various tasks are run. Each task is run by an object that implements a particular Task interface.

By combining the following tasks with those provided by Ant, you can create build scripts that compile, package, install, and test your application on the application server:

- Install and uninstall applications
- Start and stop servers in a base configuration
- Run administrative scripts or commands
- Run the Enterprise JavaBeans (EJB) deployment tool for EJB 1.x or 2.x modules
- Run the JavaServer Pages (JSP) file precompilation tool

For more detailed information about Ant, refer to the Apache organization Web site.

- To run Ant and have it automatically see the WebSphere classes, use the `ws_ant` command.

The `ws_ant` command is provided with the Apache Ant tool.

See the `app_server_root/bin/ws_ant` file for the Apache Ant tool.

- Use Ant tasks for deployment and server operation.

The Apache Ant tasks for the product reside in the Java package: `com.ibm.websphere.ant.tasks`. The API documentation for this package contains detailed information about all of the Ant tasks that are provided and how to use them.

See `com.ibm.websphere.ant.tasks` API documentation in the **Reference** section of the information center.

- Use Ant tasks for building application code.

Refer to the Rational Application Developer documentation.

- Use the Apache Struts framework to create an extensible development environment for your application, based on published standards and proven design patterns.

Apache struts is a framework that is supported by the open source community.

IBM WebSphere Application Server provides the Apache Struts JAR file in the `install_root/optionalLibraries/Apache/Struts/1.1` directory of your product installation. If using Struts in your application or Application Server, you need to configure a shared library that points to the Struts library JAR file.

The Struts framework provides the invisible underpinnings every professional Web application needs to survive. The core of Struts is a flexible control layer based on standard technologies such as Java Servlets, JavaBeans, ResourceBundles, and Extensible Markup Language (XML).

Struts encourages application architectures based on the Model 2 approach, a variation of the classic Model-View-Controller (MVC) design paradigm. Struts provides its own Controller component and integrates with other technologies to provide the Model and the View. For the Model, Struts can interact with any standard data access technology, including Enterprise Java Beans (EJB) components, and JDBC. For the View, Struts works well with JavaServer Pages (JSP) files, XSLT, or other presentation systems.

Chapter 4. Starting and stopping quick reference

Start and stop servers in your application serving environment, referring to this quick guide to the administrative clients and several other tools that are provided with this product.

- Use commands to start and stop servers.

Quick reference: Issuing commands to start and stop servers

These examples are for starting and stopping the default profile on a server. Otherwise, you might need to specify `-profileName profile_name` when invoking the command.

Application server

Run the following command. See “startServer command” on page 162 for details and variations

```
startServer server
```

where *server* is the application server that you want to start.

Stopping the servers

Use the same command as to start, except substitute stop for start. For example, to stop an application server, issue the command:

```
stopServer server
```

- Use administrative clients and tools.

Quick reference: Opening the administrative console

To open the console using the default port, enter this Web address in your Web browser:

```
http://your_fully_qualified_server_name:9060/ibm/console
```

Depending on your configuration, your Web address might differ. Other factors can affect your ability to access the console. See “Starting and logging off the administrative console” on page 32 for details, as needed.

- To launch a scripting client, see Starting the wsadmin scripting client.
 - To learn about all available administrative clients, see Chapter 2, “Using the administrative clients,” on page 31.
 - For performance monitoring, see Monitoring performance with Tivoli Performance Viewer (TPV). See the administrator commands that are listed in the **Reference** section of the information center.
- Use troubleshooting tools.
See Working with troubleshooting tools.

Chapter 5. Backing up and recovering the application serving environment

The product uses many operating system and application resources that you should consider adding to your backup and recovery procedures.

About this task

Save and restore of WebSphere Application Server resources uses the same i5/OS commands used for other i5/OS resources.

WebSphere Application Server resources can be saved while the product environment is active. When backing up database data, you may have to shut down some or all services if a snapshot cannot be obtained. This would occur if there are requests which obtain locks or have open transactions against the database being saved. In a distributed environment, you may need to consider how to get a consistent backup across several systems. If the data on systems is not closely related to data on other systems, you may be able to backup each system in isolation. If you need a snapshot across systems simultaneously, you may need to stop activity on all systems while the snapshot is taken.

How often you back up resources depends largely on when or how often you expect them to change.

- Back up your product environment configuration.

This category covers the resources that define your WebSphere Application Server operating environment. Once you have done initial setup, this information should change very infrequently. You might backup this information only when you change these settings, and not include these resources in regularly scheduled backups.

- Administrative configuration files
- Administrative configuration
- Servlet configuration files
- Security properties files
- HTTP configuration (see the documentation for your Web server)

- Back up your applications.

This category covers the applications you run using WebSphere Application Server. You should back these up the same way you back up other applications on your system. You could backup these resources every time you add or change an application, or include these resources in a regularly scheduled backup.

- Application deployment configuration files
- Administrative configuration
- Servlet source and class files
- JSP source and generated class files
- Deployed enterprise bean jar files

- Back up your application data.

This category covers the data stores used by your WebSphere Application Server applications. Unless your applications serve only static information, these resources are usually quite dynamic. You should back these up the same way you back up other business data on your system. These resources are suited for inclusion in a regularly scheduled backup.

- Servlet session data
- Enterprise bean data in a database

What to do next

If your applications are using other resources or services that are external to the product, remember to include those in your backup plan as well.

Chapter 6. Class loading

Class loaders are part of the Java virtual machine (JVM) code and are responsible for finding and loading class files. Class loaders enable applications that are deployed on servers to access repositories of available classes and resources. Application developers and deployers must consider the location of class and resource files, and the class loaders used to access those files, to make the files available to deployed applications. Class loaders affect the packaging of applications and the runtime behavior of packaged applications of deployed applications.

Before you begin

This topic describes how to configure class loaders for application files or modules that are installed on an application server.

To better understand class loaders in WebSphere Application Server, read “Class loaders.” The topic “Class loading: Resources for learning” on page 229 refers to additional sources.

About this task

Configure class loaders for application files or modules that are installed on an application server using the administrative console. You configure class loaders to ensure that deployed application files and modules can access the classes and resources that they need to run successfully.

1. If an installed application module uses a resource, create a resource provider that specifies the directory name of the resource drivers.
Do not specify the resource Java archive (JAR) file names. All JAR files in the specified directory are added into the class path of the WebSphere Application Server extensions class loader. If a resource driver requires a native library (.dll or .so file), specify the name of the directory that contains the library in the native path of the resource configuration.
2. Specify class-loader values for an application server.
3. Specify class-loader values for an installed enterprise application.
4. Specify the class-loader mode for an installed Web module.
5. If your deployed application uses shared library files, associate the shared library files with your application. Use a library reference to associate a shared library file with your application.
 - a. If you have not done so already, define shared libraries for library files that your applications need.
 - b. Define a library reference for each shared library that your application uses.

What to do next

After configuring class loaders, ensure that your application performs as desired. To diagnose and fix problems with class loaders, refer to Troubleshooting class loaders.

Class loaders

Class loaders find and load class files. Class loaders enable applications that are deployed on servers to access repositories of available classes and resources. Application developers and deployers must consider the location of class and resource files, and the class loaders used to access those files, to make the files available to deployed applications.

This topic provides the following information about class loaders in WebSphere Application Server:

- “Class loaders used and the order of use” on page 220
- “Class-loader isolation policies” on page 221
- “Class-loader modes” on page 223

Class loaders used and the order of use

The product runtime environment uses the following class loaders to find and load new classes for an application in the following order:

1. The bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine

The bootstrap class loader uses the boot class path (typically classes in `jre/lib`) to find and load classes. The extensions class loader uses the system property `java.ext.dirs` (typically `jre/lib/ext`) to find and load classes. The CLASSPATH class loader uses the CLASSPATH environment variable to find and load classes.

The CLASSPATH class loader loads the Java Platform, Enterprise Edition (Java EE) application programming interfaces (APIs) provided by the WebSphere Application Server product in the `j2ee.jar` file. Because this class loader loads the Java EE APIs, you can add libraries that depend on the Java EE APIs to the class path system property to extend a server class path. However, a preferred method of extending a server class path is to add a shared library.

2. A WebSphere extensions class loader

The WebSphere extensions class loader loads the WebSphere Application Server classes that are required at run time. The extensions class loader uses a `ws.ext.dirs` system property to determine the path that is used to load classes. Each directory in the `ws.ext.dirs` class path and every Java archive (JAR) file or ZIP file in these directories is added to the class path used by this class loader.

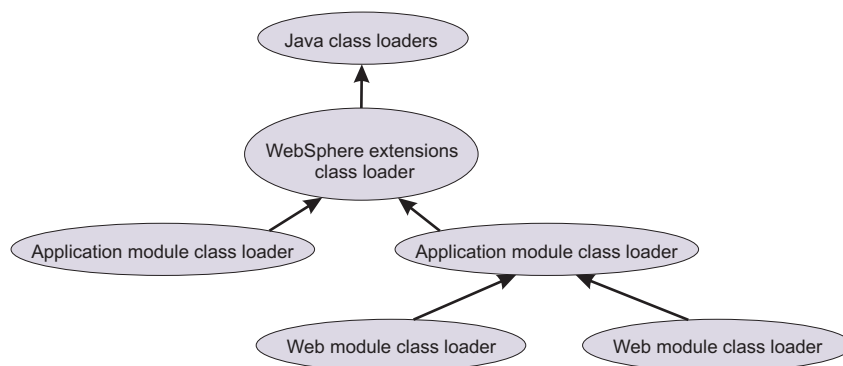
The WebSphere extensions class loader also loads resource provider classes into a server if an application module installed on the server refers to a resource that is associated with the provider and if the provider specifies the directory name of the resource drivers.

3. One or more application module class loaders that load elements of enterprise applications running in the server

The application elements can be Web modules, enterprise bean (EJB) modules, resource adapter archives (RAR files), and dependency JAR files. Application class loaders follow Java EE class-loading rules to load classes and JAR files from an enterprise application. The product enables you to associate shared libraries with an application.

4. Zero or more Web module class loaders

By default, Web module class loaders load the contents of the `WEB-INF/classes` and `WEB-INF/lib` directories. Web module class loaders are children of application class loaders. You can specify that an application class loader load the contents of a Web module rather than the Web module class loader.



Each class loader is a child of the previous class loader. That is, the application module class loaders are children of the WebSphere extensions class loader, which is a child of the CLASSPATH Java class loader. Whenever a class needs to be loaded, the class loader usually delegates the request to its parent class loader. If none of the parent class loaders can find the class, the original class loader attempts to load the class. Requests can only go to a parent class loader; they cannot go to a child class loader. If the WebSphere extensions class loader is requested to find a class in a Java EE module, it cannot go to the application module class loader to find that class and a `ClassNotFoundException` error occurs. After a

class is loaded by a class loader, any new classes that it tries to load reuse the same class loader or go up the precedence list until the class is found.

Class-loader isolation policies

The number and function of the application module class loaders depend on the class-loader policies that are specified in the server configuration. Class loaders provide multiple options for isolating applications and modules to enable different application packaging schemes to run on an application server.

Two class-loader policies control the isolation of applications and modules:

Class-loader policy	Description
Application	Application class loaders load EJB modules, dependency JAR files, embedded resource adapters, and application-scoped shared libraries. Depending on the application class-loader policy, an application class loader can be shared by multiple applications (Single) or unique for each application (Multiple). The application class-loader policy controls the isolation of applications that are running in the system. When set to Single , applications are not isolated. When set to Multiple , applications are isolated from each other.
WAR	<p>By default, Web module class loaders load the contents of the <code>WEB-INF/classes</code> and <code>WEB-INF/lib</code> directories. The application class loader is the parent of the Web module class loader. You can change the default behavior by changing the Web application archive (WAR) class-loader policy of the application.</p> <p>The WAR class-loader policy controls the isolation of Web modules. If this policy is set to Application, then the Web module contents also are loaded by the application class loader (in addition to the EJB files, RAR files, dependency JAR files, and shared libraries). If the policy is set to Module, then each Web module receives its own class loader whose parent is the application class loader.</p> <p>Note: The console and the underlying <code>deployment.xml</code> file use different names for WAR class-loader policy values. In the console, the WAR class-loader policy values are Application or Module. However, in the underlying <code>deployment.xml</code> file where the policy is set, the WAR class-loader policy values are Single instead of Application, or Multiple instead of Module. Application is the same as Single, and Module is the same as Multiple.</p>

Note: WebSphere Application Server class loaders never load application client modules.

For each application server in the system, you can set the application class-loader policy to **Single** or **Multiple**. When the application class-loader policy is set to **Single**, then a single application class loader loads all EJB modules, dependency JAR files, and shared libraries in the system. When the application class-loader policy is set to **Multiple**, then each application receives its own class loader that is used for loading the EJB modules, dependency JAR files, and shared libraries for that application.

An application class loader loads classes from Web modules if the application's WAR class-loader policy is set to **Application**. If the application's WAR class-loader policy is set to **Module**, then each WAR module receives its own class loader.

The following example shows that when the application class-loader policy is set to **Single**, a single application class loader loads all of the EJB modules, dependency JAR files, and shared libraries of all applications on the server. The single application class loader can also load Web modules if an application has its WAR class-loader policy set to **Application**. Applications that have a WAR class-loader policy set to **Module** use a separate class loader for Web modules.

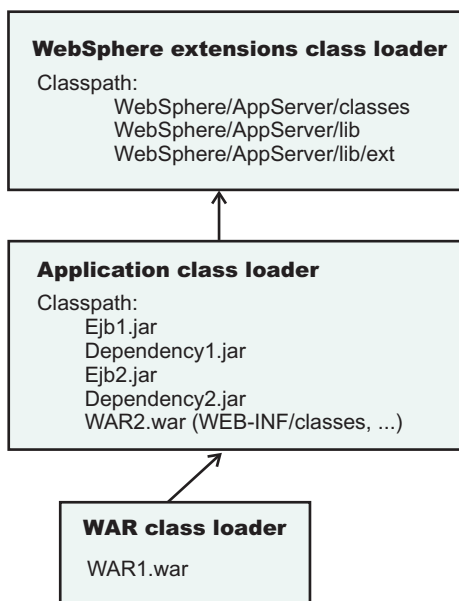
```
Server's application class-loader policy: Single
Application's WAR class-loader policy: Module
```

```
Application 1
```

```

Module: EJB1.jar
Module: WAR1.war
  MANIFEST Class-Path: Dependency1.jar
  WAR Classloader Policy = Module
Application 2
Module: EJB2.jar
  MANIFEST Class-Path: Dependency2.jar
Module: WAR2.war
  WAR Classloader Policy = Application

```



The following example shows that when the application class-loader policy of an application server is set to `Multiple`, each application on the server has its own class loader. An application class loader also loads its Web modules if the application WAR class-loader policy is set to `Application`. If the policy is set to `Module`, then a Web module uses its own class loader.

```

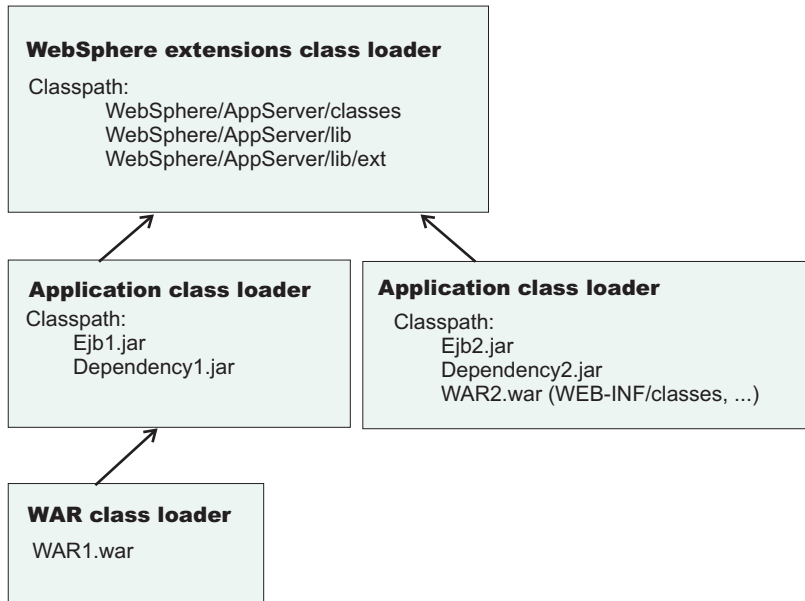
Server's application class-loader policy: Multiple
Application's WAR class-loader policy: Module

```

```

Application 1
Module: EJB1.jar
Module: WAR1.war
  MANIFEST Class-Path: Dependency1.jar
  WAR Classloader Policy = Module
Application 2
Module: EJB2.jar
  MANIFEST Class-Path: Dependency2.jar
Module: WAR2.war
  WAR Classloader Policy = Application

```



Class-loader modes

The class-loader delegation mode, also known as the *class loader order*, determines whether a class loader delegates the loading of classes to the parent class loader. The following values for class-loader mode are supported:

Class-loader mode	Description
Parent first Also known as Classes loaded with parent class loader first.	The Parent first class-loader mode causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local class path. This value is the default for the class-loader policy and for standard JVM class loaders.
Parent last Also known as Classes loaded with local class loader first or Application first.	The Parent last class-loader mode causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent. Using this policy, an application class loader can override and provide its own version of a class that exists in the parent class loader.

The following settings determine the mode of a class loader:

- If the application class-loader policy of an application server is `Single`, the server-level mode value defines the mode for an application class loader.
- If the application class-loader policy of an application server is `Multiple`, the application-level mode value defines the mode for an application class loader.
- If the WAR class-loader policy of an application is `Module`, the module-level mode value defines the mode for a WAR class loader.

Configuring class loaders of a server

You can configure the application class loaders for an application server. Class loaders enable applications that are deployed on the application server to access repositories of available classes and resources.

Before you begin

This topic assumes that an administrator created an application server on a WebSphere Application Server product.

About this task

Configure the class loaders of an application server to set class-loader policy and mode values which affect all applications that are deployed on the server. Use the administrative console to configure the class loaders.

1. Click **Servers** → **Server Types** → **WebSphere application servers** → **server_name** to access an application server settings page.
2. Specify the application class-loader policy for the application server.

The application class-loader policy controls the isolation of applications that run in the system (on the server). An application class loader groups enterprise bean (EJB) modules, shared libraries, resource adapter archives (RAR files), and dependency Java archive (JAR) files associated to an application. Dependency JAR files are JAR files that contain code which can be used by both enterprise beans and servlets. The application class-loader policy controls whether an application class loader can be shared by multiple applications or is unique for each application.

Use the application server settings page to specify the application class-loader policy for the server:

Option	Description
Single	Applications are not isolated from each other. Uses a single application class loader to load all of the EJB modules, shared libraries, and dependency JAR files in the system.
Multiple	Applications are isolated from each other. Gives each application its own class loader to load the EJB modules, shared libraries, and dependency JAR files of that application.

3. Specify the application class-loader mode for the application server.
The application class loading mode specifies the class-loader mode when the application class-loader policy is **Single**.

On the application server settings page, select either of the following values:

Option	Description
Classes loaded with parent class loader first	Causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local class path. Classes loaded with parent class loader first is the default value for class loading mode. This value is also known as parent first.
Classes loaded with local class loader first (parent last)	Causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent. Using this policy, an application class loader can override and provide its own version of a class that exists in the parent class loader.

4. Specify the class-loader mode for the class loader.
 - a. On the application server settings page, click **Java and Process Management** → **Class loader** to access the Class loader page.
 - b. On the Class loader page, click **New** to access the settings page for a class loader.

- c. On the class loader settings page, specify the class loader order.

The `Classes loaded with parent class loader first` value causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local class path.

The `Classes loaded with local class loader first (parent last)` value causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent.

- d. Click **OK**.

An identifier is assigned to a class-loader instance. The instance is added to the collection of class loaders shown on the Class loader page.

What to do next

Save the changes to the administrative configuration.

Class loader collection

Use this page to manage class-loader instances on an application server. A class loader determines whether an application class loader or a parent class loader finds and loads Java class files for an application.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → *server_name* → **Java and Process Management** → **Class loader**.

Class loader ID

Specifies a string that is unique to the server identifying the class-loader instance. The product assigns the identifier.

Class loader order

Specifies whether the class loader searches in the parent class loader or in the application class loader first to load a class. The standard for development kit class loaders and WebSphere Application Server class loaders is `Classes loaded with parent class loader first (Parent first)`. By specifying `Classes loaded with local class loader first (Parent last)`, your application can override classes contained in the parent class loader, but this action can potentially result in `ClassCastException` or `LinkageErrors` if you have mixed use of overridden classes and non-overridden classes.

Class loader settings

Use this page to configure a class loader for applications that reside on an application server.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → *server_name* → **Java and Process Management** → **Class loader** → *class_loader_ID*.

Class loader ID

Specifies a string that is unique to the server identifying the class-loader instance. The product assigns the identifier.

Data type String

Class loader order

Specifies whether the class loader searches in the parent class loader or in the application class loader first to load a class. The standard for development kit class loaders and WebSphere Application Server class loaders is `Classes loaded with parent class loader first`. By specifying `Classes loaded with local class loader first (parent last)`, your application can override classes contained in the parent

class loader, but this action can potentially result in `ClassCastException` or `LinkageErrors` if you have mixed use of overridden classes and non-overridden classes.

The options are `Classes loaded with parent class loader first` and `Classes loaded with local class loader first (parent last)`. The default is to search in the parent class loader before searching in the application class loader to load a class.

For your application to use the default configuration of Jakarta Commons Logging in this product, set this application class loader order to `Classes loaded with parent class loader first`. For your application to override the default configuration of Jakarta Commons Logging, your application must provide the configuration in a form supported by Jakarta Commons Logging and this class loader order must be set to `Classes loaded with local class loader first (parent last)`. Also, to override the default configuration, set the class loader order for each Web module in your application so that the correct logger factory loads.

Data type	String
Default	Parent first

Configuring application class loaders

You can set values that control the class-loading behavior of an installed enterprise application. Class loaders enable an application to access repositories of available classes and resources.

Before you begin

This topic assumes that you installed an application on an application server.

About this task

Configure the class loaders of an enterprise application to set class-loader policy and mode values for this application.

An application class loader groups enterprise bean (EJB) modules, shared libraries, resource adapter archives (RAR files), and dependency Java archive (JAR) files associated to an application. Dependency JAR files are JAR files that contain code which can be used by both enterprise beans and servlets.

An application class loader is the parent of a Web application archive (WAR) class loader. By default, a Web module has its own WAR class loader to load the contents of the Web module. The WAR class-loader policy value of an application class loader determines whether the WAR class loader or the application class loader is used to load the contents of the Web module.

Use the administrative console to configure the class loaders.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Class loading and update detection** to access the settings page for an application class loader.
2. Specify whether to reload application classes when the application or its files are updated.
By default, class reloading is not enabled. Select **Override class reloading settings for Web and EJB modules** to choose to reload application classes. You might specify different values for EJB modules and for Web modules such as servlets and JavaServer Pages (JSP) files.

- Specify the number of seconds to scan the application's file system for updated files.
The value specified for **Polling interval for updated files** takes effect only if class reloading is enabled. The default is the value of the reloading interval attribute in the IBM extension (META-INF/ibm-application-ext.xml) file of the enterprise application (EAR file). You might specify different values for EJB modules and for Web modules such as servlets and JSP files.

To enable reloading, specify an integer value that is greater than zero (for example, 1 to 2147483647).

To disable reloading, specify zero (0).

- Specify the class loader order for the application.

The application class loader order specifies whether the class loader searches in the parent class loader or in the application class loader first to load a class. The default is to search in the parent class loader before searching in the application class loader to load a class.

Select either of the following values for **Classes loader order**:

Option	Description
Classes loaded with parent class loader first	Causes the class loader to search in the parent class loader first to load a class. This value is the standard for Development Kit class loaders and WebSphere Application Server class loaders.
Classes loaded with local class loader first (parent last)	Causes the class loader to search in the application class loader first to load a class. By specifying <code>Classes loaded with local class loader first (parent last)</code> , your application can override classes contained in the parent class loader. Note: Specifying the <code>Classes loaded with local class loader first (parent last)</code> value might result in <code>LinkageErrors</code> or <code>ClassCastException</code> messages if you have mixed use of overridden classes and non-overridden classes.

- Specify whether to use a single or multiple class loaders to load Web application archives (WAR files) of your application.

By default, Web modules have their own WAR class loader to load the contents of the `WEB-INF/classes` and `WEB-INF/lib` directories. The default WAR class loader value is `Class loader for each WAR file in application`, which uses a separate class loader to load each WAR file. Setting the value to `Single class loader for application` causes the application class loader to load the Web module contents as well as the EJB modules, shared libraries, RAR files, and dependency JAR files associated to the application. The application class loader is the parent of the WAR class loader.

Select either of the following values for **WAR class loader policy**:

Option	Description
Class loader for each WAR file in application	Uses a different class loader for each WAR file.
Single class loader for application	Uses a single class loader to load all of the WAR files in your application.

- Click **OK**.

What to do next

Save the changes to the administrative configuration.

Configuring Web module class loaders

You can set values that control the class-loading behavior of an installed Web module.

Before you begin

This topic assumes that you installed a Web module on an application server.

About this task

Configure the class loader order value of an installed Web module. By default, a Web module has its own Web application archive (WAR) class loader to load the contents of the Web module, which are in the WEB-INF/classes and WEB-INF/lib directories.

An application class loader is the parent of a WAR class loader. The WAR class-loader policy value of an application class loader determines whether the WAR class loader or the application class loader is used to load the contents of the Web module.

The default WAR class loader policy value is `Class loader` for each WAR file in application. If the policy is set to `Class loader` for each WAR file in application, then each Web module receives its own class loader whose parent is the application class loader. If the policy is set to `Single class loader` for application on the settings page of an application class loader, then the application class loader loads the Web module contents as well as the enterprise bean (EJB) modules, shared libraries, resource adapter archives (RAR files), and dependency Java archive (JAR) files associated to an application. Thus, the configuration of the parent application class loader affects the WAR class loader.

Use the administrative console to configure the application and WAR class loaders.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

1. If you have not done so already, configure the application class loader.

Settings such as **Override class reloading settings for Web and EJB modules**, **Polling interval for updated files** and **WAR class loader policy** can affect Web module class loading.

If **WAR class loader policy** is set to `Class loader` for each WAR file in application, then the Web module receives its own class loader and the WAR class-loader policy of the Web module defines the mode for a WAR class loader. If the policy is set to `Single class loader` for application, then the application class loader loads the Web module contents.

2. Specify the class loader order for the installed Web module.

The Web module class-loader mode specifies whether the class loader searches in the parent application class loader or in the WAR class loader first to load a class. The default is to search in the parent application class loader before searching in the WAR class loader to load a class.

Select either of the following values for **Class loader order**:

Option	Description
Classes loaded with parent class loader first	Causes the class loader to search in the parent application class loader first to load a class. This is the standard for Development Kit class loaders and WebSphere Application Server class loaders. Note: If classes and resources needed by the Web module cannot be accessed by the application class loader, but can be accessed by the WAR class loader, specify <code>Classes loaded with local class loader first (parent last)</code> . If the application class loader cannot find a class, the class loader delegates the request to find the class to its parent, the WebSphere Application Server extensions class loader. If the WebSphere Application Server extensions class loader cannot find the class, the class loader delegates the request to its parent, the bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine. Requests can only go to a parent class loader; they cannot go to a child class loader. Thus, if <code>Classes loaded with parent class loader first</code> is specified, the WAR class loader never receives a request to load a class.
Classes loaded with local class loader first (parent last)	Causes the class loader to search in the WAR class loader first to load a class. By specifying <code>Classes loaded with local class loader first (parent last)</code> , your WAR class loader can override classes contained in the parent application class loader. Note: Specifying the <code>Classes loaded with local class loader first (parent last)</code> value might result in <code>LinkageErrors</code> or <code>ClassCastException</code> messages if you have mixed use of overridden classes and non-overridden classes.

3. Click **OK**.

What to do next

Save the changes to the administrative configuration.

Class loading: Resources for learning

Additional information and guidance on class loading is available on various Internet sites.

Use the following links to find relevant supplemental information about class loaders. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- “Programming model and decisions” on page 230
- “Programming instructions and examples” on page 230
- “Programming specifications” on page 230

Programming model and decisions

- Demystifying class loading problems, Part 1: An introduction to class loading and debugging tools - Learn how class loading works and how your JVM can help you sort out class loading problems (*developerWorks*, November 2005), http://www.ibm.com/developerworks/java/library/j-dclp1/?S_TACT=106AH10W&S_CMP=NC
- Demystifying class loading problems, Part 2: Basic class loading exceptions - An in-depth look at some simple class loading quirks and conundrums (*developerWorks*, December 2005), http://www.ibm.com/developerworks/java/library/j-dclp2.html?S_TACT=105AGX10&S_CMP=NC
- Demystifying class loading problems, Part 3: Tackling more unusual class loading problems - Understand class loading and quash subtle exceptions (*developerWorks*, December 2005), http://www.ibm.com/developerworks/java/library/j-dclp3/?S_TACT=105AGX10&S_CMP=NC
- J2EE Class Loading Demystified (*developerWorks*, August 2002), http://www.ibm.com/developerworks/websphere/library/techarticles/0112_deboer/deboer.html
- Java programming dynamics, Part 1: Classes and class loading - A look at classes and what goes on as they're loaded by a JVM (*developerWorks*, April 2003), <http://www.ibm.com/developerworks/java/library/j-dyn0429/>

Programming instructions and examples

- *WebSphere Application Server V6.1: System Management Configuration Handbook*, SG24-7304-00, <http://www.redbooks.ibm.com/abstracts/SG247304.html?Open>
- *IBM WebSphere Developer Technical Journal: Co-hosting multiple versions of J2EE applications*, http://www.ibm.com/developerworks/websphere/techjournal/0405_poddar/0405_poddar.html

Programming specifications

- Specifications and API documentation

Chapter 7. Deploying and administering enterprise applications

Deploying an enterprise application file consists of installing an application file on a server configured to hold installable Java Platform, Enterprise Edition (Java EE) modules.

Before you begin

Before installing an enterprise application or other installable module on an application server, you must develop the module, assemble the module, and configure the target server. Before choosing a deployment target for the module, ensure that the target version is compatible with your module.

About this task

During installation, you can configure the module enough to enable it to run on the server. After installation, you can configure the module further, start or stop the application, and otherwise manage its activity.

The topics in this section describe how to deploy and administer applications or modules using the administrative console. You can also use scripting or administrative programs (JMX).

- Install Java EE application files on an application server.
- Edit the administrative configuration for an application.
- Optional: View the deployment descriptor for an application or module.
- Start and stop enterprise applications.
- Export enterprise applications.
- Export a file in a Java EE application or module.
- Export DDL files.
- Update a Java EE application or module.
- Uninstall enterprise applications.
- Remove a file from a Java EE application or module.

What to do next

After making changes to administrative configurations of your applications in the administrative console, ensure that you save the changes.

Enterprise (Java EE) applications

Enterprise applications (or Java EE applications) are applications that conform to the Java Platform, Enterprise Edition (Java EE) specification. Prior to Java EE 5, the specification name was Java 2 Platform, Enterprise Edition (J2EE). The term *Java EE* includes Java EE 5 and J2EE specifications.

Enterprise applications can consist of the following:

- Zero or more EJB modules (packaged in JAR files)
- Zero or more Web modules (packaged in WAR files)
- Zero or more connector modules (packaged in RAR files)
- Zero or more Session Initiation Protocol (SIP) modules (packaged in SAR files)
- Zero or more application client modules
- Additional JAR files containing dependent classes or other components required by the application
- Any combination of the above

A Java EE application is represented by, and packaged in, an enterprise archive (EAR) file.

System applications

A *system application* is a Java Platform, Enterprise Edition (Java EE) enterprise application that is central to a WebSphere Application Server product.

Examples of system applications include *isclite*, *managementEJB* and *filetransfer*.

Because a system application is an important part of a WebSphere Application Server product, a system application is deployed when the product is installed and is updated only through a product fix or upgrade. For some system applications, such as *filetransfer*, users cannot change the metadata for the system application, unless the metadata assigns users and groups for security purposes. For these applications, non-security related metadata such as its Java EE bindings or extensions must be updated through a product fix or upgrade.

System applications are not shown in the list of installed applications on the console Enterprise Applications page, or through wsadmin and Java application programming interfaces, to prevent users from accidentally stopping, updating or removing the system applications.

Note that Java EE Samples are not system applications even though they are provided as part of a WebSphere Application Server product. Similarly, applications that support changes to their metadata are not system applications.

Common deployment framework

The *common deployment framework* enables you to implement plug-ins that add steps to default Java Platform, Enterprise Edition (Java EE) application management operations such as install, uninstall, edit and update.

Using the framework, you can implement management operations on specific types of deployable contents. For example, the deployable contents might include EAR, WAR, JAR or other Java EE modules and the management operations might include install and uninstall. Each operation is divided into a number of steps. For example, the install operation has steps for EJBDeploy and JavaServer Pages (JSP) compilation, among others. Using the common deployment framework, you can add steps to the default logic for Java EE operations.

The product supports framework plug-ins that extend deployment of EAR files. An EAR file has operations such as `createEarWrapper`, `installApplication`, `uninstallApplication` and `editApplication`. Using a framework plug-in, you can add steps to default install operations that support, for example, creating additional configuration artifacts in a configuration session, modifying an input EAR file using code generation, or additional validating of input parameters.

To extend application management operations using the framework, a plug-in must do the following:

- Implement each step.

A *step* runs logic that performs an operation. A step can access the deployment context and the deployable object. The *deployment context* provides information such as the operation name, the configuration session identifier, the temporary location for creating temporary files, operations parameters, and the like. A step is added by the extension provider.

- Implement an extension provider that adds each implemented step.

An *extension provider* is a class that provides steps for an operation on a given type, the EAR file type.

- Register the plug-in with a WebSphere Application Server server.

The plug-in is implemented as an Eclipse plug-in and is placed in `app_server_root/plugins` directory. Add the extension point for the extension provider in the `META-INF/plugin.xml` file within the plug-in JAR file.

For an example of these steps, refer to “Extending application management operations through programming” on page 355.

Installing enterprise application files

As part of deploying an application, you install application files on a server configured to hold installable modules.

Before you begin

Before you can install your Java Platform, Enterprise Edition (Java EE) application files on an application server, you must assemble modules as needed.

Also, before you install the files, configure the target application server. As part of configuring the server, determine whether your application files can be installed to your deployment targets.

About this task

You can install the following modules on a server:

- Enterprise archive (EAR)
- Enterprise bean (EJB)
- Web archive (WAR)
- Session Initiation Protocol (SIP) module (SAR)
- Resource adapter (connector or RAR)
- Application client modules

Application client files can be installed in a WebSphere Application Server configuration but cannot be run on a server.

Complete the following steps to install your files.

1. Determine which method to use to install your application files. The product provides several ways to install modules.
2. Install the application files using
 - Administrative console
 - wsadmin scripts
 - Java administrative programs that use Java Management Extensions (JMX) application programming interfaces (APIs)
 - Java programs that define a Java EE DeploymentManager object in accordance with Java EE Application Deployment specification (JSR-88)
3. Start the deployed application files using
 - Administrative console
 - wsadmin startApplication
 - Java programs that use ApplicationManager or AppManagement MBeans
 - Java programs that define a Java EE DeploymentManager object in accordance with Java EE Application Deployment specification (JSR-88)

What to do next

Save the changes to your administrative configuration.

Next, test the application. For example, point a Web browser at the URL for a deployed application (typically `http://hostname:9060/Web_module_name`, where *hostname* is your valid Web server and 9060 is the default port number) and examine the performance of the application. If the application does not perform as desired, edit the application configuration, then save and test it again.

Installable enterprise module versions

The contents of a Java Platform, Enterprise Edition (Java EE) module affect whether you can install the module on a deployment target. A *deployment target* is a server on a WebSphere Application Server product.

Installable application modules

Select only appropriate deployment targets for a module. You must install an application, enterprise bean (EJB) module, Session Initiation Protocol (SIP) module (SAR), or Web module on a Version 7.x target under any of the following conditions:

- The module supports Java Platform, Enterprise Edition (Java EE) 5.
- The module calls a 7.x runtime application programming interface (API).
- The module uses a 7.x product feature.

If a module supports Java 2 Platform, Enterprise Edition (J2EE) 1.4, then you must install the module on a Version 6.x or 7.x deployment target. Modules that call a 6.1.x API or use a 6.1.x feature can be installed on a 6.1.x or 7.x deployment target. Modules that call a 6.0.x API or use a 6.0.x feature can be installed on a 6.0.x, 6.1.x or 7.x deployment target. Modules that require 6.1.x feature pack functionality can be installed on a 7.x deployment target or on a 6.1.x deployment target that has been enabled with that feature pack.

Selecting options such as **Precompile JavaServer Pages files**, **Use binary configuration**, **Deploy Web services** or **Deploy enterprise beans** during application installation indicates that the application uses 6.1.x product features. You cannot deploy such applications on a 5.x or 6.0.x deployment target. You must deploy such applications on a 6.1.x or 7.x deployment target.

You can install an application or module developed for a Version 5.x product on a 5.x, 6.x or 7.x deployment target.

Note: You must package container-managed persistence (CMP) or bean-managed persistence (BMP) entity beans in an EJB 2.1 or earlier module. You cannot install an EJB 3.0 module that contains CMP or BMP entity beans. Installation fails when a CMP or BMP entity bean is packaged in an EJB 3.0 module. You can install EJB 2.1 or earlier modules on a 5.x, 6.x or 7.x deployment target.

Installable RAR files

You can install a standalone resource adapter (connector) module, or RAR file, developed for a Version 5.x product to a 5.x, 6.x or 7.x deployment target, provided the module does not call any 6.x or 7.x runtime APIs. If the module calls a 6.x API, then you must install the module on a 6.x or 7.x deployment target. If the module calls a 7.x API, then you must install the module on a 7.x deployment target.

Deployment targets

A *5.x deployment target* is a server on a WebSphere Application Server Version 5 product.

A *6.x deployment target* is a server on a WebSphere Application Server Version 6 product.

A *7.x deployment target* is a server on a WebSphere Application Server Version 7 product.

Table 3. Compatible deployment target versions for 5.x, 6.x and 7.x modules

Module type	Module Java support	Module calls 6.x or 7.x runtime APIs or uses 6.x or 7.x features?	Client versions that can install module	Deployment target versions

Table 3. Compatible deployment target versions for 5.x, 6.x and 7.x modules (continued)

Application, EJB, Web, or client	J2EE 1.3	No	5.x, 6.x or 7.x	5.x, 6.x or 7.x
Application, EJB, Web, or client	J2EE 1.3	Yes	6.x for 6.x or 7.x APIs or features 7.x for 7.x APIs or features	6.x or 7.x You must install modules that call 6.1.x runtime APIs or use 6.1.x features on a 6.1.x or 7.x deployment target. You can install modules that call 6.0.x runtime APIs or use 6.0.x features on any 6.x or 7.x deployment target.
Application, EJB, SAR, Web, or client	J2EE 1.4	Yes or No	6.x or 7.x	6.x or 7.x
Application, EJB, SAR, Web, or client	Java EE 5	Yes or No	7.x	7.x
Resource adapter	JCA 1.0	No	5.x, 6.x or 7.x	5.x, 6.x or 7.x
Resource adapter	JCA 1.0	Yes	6.x or 7.x	6.x or 7.x You must install modules that call 6.1.x runtime APIs on a 6.1.x or 7.x deployment target. You can install modules that call 6.0.x runtime APIs on any 6.x or 7.x deployment target.
Resource adapter	JCA 1.5	Yes or No	6.x or 7.x	6.x or 7.x You must install modules that call 6.1.x runtime APIs on a 6.1.x or 7.x deployment target. You can install modules that call 6.0.x runtime APIs on any 6.x or 7.x deployment target.

Ways to install enterprise applications or modules

The product provides several ways to install Java Platform, Enterprise Edition (Java EE) application files.

Installable files include enterprise archive (EAR), enterprise bean (EJB), Web archive (WAR), Session Initiation Protocol (SIP) module (SAR), resource adapter (connector or RAR), and application client modules. They can be installed on a server. Application client files can be installed in a WebSphere Application Server configuration but cannot be run on a server.

Table 4. Ways to install application files

Option	Method	Modules	Comments	Starting after install
Administrative console install wizard See “Installing enterprise application files with the console” on page 238.	Click Applications → New application → New Enterprise Application in the console navigation tree and follow instructions in the wizard.	Files for all of the following modules: <ul style="list-style-type: none"> • EAR • EJB • WAR • SAR • RAR • Application client 	Provides one of the easier ways to install application files. See “Preparing for application installation settings” on page 244 for guidance. For applications that do not require changes to the default bindings, after you specify the application file, expand Choose to generate default bindings and mappings , select Generate default bindings , click the Summary step, and then click Finish .	Click Start on the Enterprise applications page accessed by clicking Applications → Application Types → WebSphere enterprise applications in the console navigation tree.
wsadmin scripts	Invoke AdminApp object install commands in a script or at a command prompt.	Files for all of the following modules: <ul style="list-style-type: none"> • EAR • EJB • WAR • SAR • RAR • Application client 	“Getting started with scripting” in the Using the administrative clients PDF provides an overview of wsadmin.	<ul style="list-style-type: none"> • Invoke the AdminApp startApplication command. • Invoke the startApplication method on an ApplicationManager MBean using AdminControl.
Java application programming interfaces	Install programs by completing the steps in “Installing an application through programming” in the Using the administrative clients PDF.	All EAR files	Use MBeans to install the application. “Managing applications through programming” in the Using the administrative clients PDF provides an overview of Java MBean programming.	Start the application by calling the startApplication method on a proxy.

Table 4. Ways to install application files (continued)

Option	Method	Modules	Comments	Starting after install
Rapid deployment tools Refer to topics under Rapid deployment of J2EE applications.	Briefly, do the following: 1. Update your J2EE application files. 2. Set up the rapid deployment environment. 3. Create a free-form project. 4. Launch a rapid deployment session. 5. Drop your updated application files into the free-form project.	J2EE modules at the J2EE 1.3 or 1.4 specification levels, including EAR files and the following stand-alone modules: <ul style="list-style-type: none"> • EJB • WAR • SAR • RAR • Application client The rapid deployment tools do not support the Java EE 5.0 or J2EE 1.2 specification levels.	Rapid deployment tools offer the following advantages: <ul style="list-style-type: none"> • You do not need to assemble your J2EE application files prior to deployment. • You do not need to use other installation tools mentioned in this table to deploy the files. 	Use any of the above options to start the application. Clicking Start on the Enterprise applications page is the easiest option.
Java programs	Code programs that use Java EE DeploymentManager (JSR-88) methods.	All Java EE modules, including EAR files and the following stand-alone modules: <ul style="list-style-type: none"> • EJB • WAR • SAR • RAR • Application client 	<ul style="list-style-type: none"> • Uses Java EE Application Deployment Specification (JSR-88). • Can customize modules using DConfigBeans. 	Call the Java EE DeploymentManager (JSR-88) start method in a program to start the deployed modules when the module's running environment initializes.

Note: In the Version 6.1 Feature Pack for Web services and Feature Pack for EJB 3.0, the default is to scan pre-Java EE 5 Web application modules to identify JAX-WS services and to scan pre-Java EE 5 Web application modules and EJB modules for service clients during application installation. For Version 7.0, the default is not to scan pre-Java EE 5 modules for annotations during application installation or server startup. To preserve backward compatibility with either or both feature packs, you can define Java virtual machine custom properties on servers to request scanning during application installation and server startup.

- You can define these custom properties using the console. Click **Servers** → **Server Types** → **WebSphere application servers** → *server name* → **Java and Process Management** → **Process definition** → **Java virtual machine** → **Custom properties**. To request scanning for Feature Pack for Web services modules, set the `com.ibm.websphere.webservices.UseWSFEP61ScanPolicy` custom property to `true`. To request scanning for Feature Pack for EJB 3.0 modules, set the `com.ibm.websphere.ejb.UseEJB61FEPScanPolicy` custom property to `true`. The default value for each of these custom properties is `false`. You must change the setting on each server that requires a change in the default.
- You can specify values for these custom properties in the META-INF/MANIFEST.MF file of a module. Values specified in the META-INF/MANIFEST.MF file always take precedence over a server-level setting.
- When using the `launchClient` tool to run the application client, you need to define these custom properties using the `-CCD` option. For example, `launchClient app.ear -CCD-CCDcom.ibm.websphere.ejb.UseEJB61FEPScanPolicy=true`.

Installing enterprise application files with the console

Installing Java Platform, Enterprise Edition (Java EE) application files consists of placing assembled enterprise application, Web, enterprise bean (EJB), or other installable modules on a server or cluster configured to hold the files. Installed files that start and run properly are considered *deployed*.

Before you begin

Before installing enterprise application files, ensure that you are installing your application files onto a compatible deployment target. If the deployment target is not compatible, select a different target.

Optionally, determine whether the application that you are installing uses library files that other deployed applications also use. You can define a shared library for each of these shared files. Using shared libraries reduces the number of library file copies on your workstation or server.

About this task

To install new enterprise application files to a WebSphere Application Server configuration, you can use the following options:

- Administrative console
- wsadmin scripts
- Java MBean programs
- Java programs that call Java EE DeploymentManager (JSR-88) methods

This topic describes how to use the administrative console to install an application, EJB component, Session Initiation Protocol (SIP) module (SAR), or Web module.

Note: After you start performing the steps below, click **Cancel** to exit if you decide not to install the application. Do not simply move to another administrative console page without first clicking **Cancel** on an application installation page.

1. Click **Applications** → **New application** → **New Enterprise Application** in the console navigation tree.
2. On the first Preparing for application installation page:

- a. Specify the full path name of the source enterprise application file (.ear file otherwise known as an *EAR file*).

The EAR file that you are installing can be either on the client machine (the machine that runs the Web browser) or on the server machine (the machine to which the client is connected). If you specify an EAR file on the client machine, then the administrative console uploads the EAR file to the machine on which the console is running and proceeds with application installation.

You can also specify a stand-alone Web application archive (WAR), SAR, or Java archive (JAR) file for installation.

If the EAR file resides on the server machine, and the server is an iSeries server, ensure that user profile QEJBSVR has *R authority to the EAR file and at least *X authority to all the directories in the path containing the EAR file.

- b. Click **Next**.

3. On the second Preparing for application installation page:

- a. Select whether to view all installation options.

Fast Path - Prompt only when additional information is required

Displays the module mapping step as well as any steps that require you to specify needed information to install the application successfully.

Detailed - Show all installation options and parameters

Displays all installation options.

- b. Select whether to generate default bindings.

Select **Generate default bindings** to have the product supply default values for incomplete Java Naming and Directory (JNDI) and other bindings in the application. The product does not change existing bindings.

You do not need to specify JNDI values for EJB bean, local home, remote home, or business interfaces of EJB 3.0 modules. The product assigns container default values during run time. Similarly, for any EJB reference within an EJB 3.0, Web 2.4, or Web 2.5 module, you do not need to specify JNDI values because the product resolves the targets automatically during run time. Even when you select **Generate default bindings**, the product does not generate default values for those JNDI values but it does generate default values for other bindings such as virtual host.

You can customize default values used in generating default bindings. “Preparing for application installation binding settings” on page 244 describes available customizations and provides sample bindings.

- c. Click **Next**. If security warnings are displayed, click **Continue**. The Install New Application pages are displayed. If you chose to generate default bindings, you can proceed to the Summary step. “Example: Installing an EAR file using the default bindings” on page 262 provides sample steps.
4. Specify values for installation options as needed.

You can click on a step number to move directly to that panel instead of clicking **Next**. The contents of the application or module that you are installing determines which panels are available.

Panel	Description
Select installation options	On the Select installation options panel, provide values for the settings specific to the product. Default values are used if you do not specify a value.
Map modules to servers	On the Map modules to servers panel, specify deployment targets where you want to install the modules contained in your application. Modules can be installed on the same deployment target or dispersed among several deployment targets. Each module must be mapped to a target server. On single-server products, a deployment target can be an application server or Web server.
Provide options to compile JSPs	If the Precompile JavaServer Pages files setting is enabled on the Select installation options panel and your application uses JavaServer Pages (JSP) files, then you can specify JSP compiler options on the Provide options to compile JSPs panel.
Provide JNDI names for beans	On the Provide JNDI names for beans panel, specify a JNDI name for each enterprise bean in every EJB 2.1 and earlier module. You must specify a JNDI name for every enterprise bean defined in the application. For example, for the EJB module MyBean.jar, specify MyBean. As to EJB 3.0 modules, you can specify JNDI names, local home JNDI names, remote home JNDI names, or no JNDI names. If you do not specify a value, the product provides a default value.
Bind EJB business	On the Bind EJB business panel, you can specify business interface JNDI names for EJB 3.0 modules. If you specified a JNDI name for a bean on the Provide JNDI names for beans panel, do not specify a business interface JNDI name on this panel for the same bean. If you do not specify the JNDI name for a bean, you can optionally specify a business interface JNDI name. When you do not specify a business interface JNDI name, the product provides a container default.
Map default data sources for modules containing 1.x entity beans	If your application uses EJB modules that contain Container Managed Persistence (CMP) beans that are based on the EJB 1.x specification, for Map default data sources for modules containing 1.x entity beans , specify a JNDI name for the default data source for the EJB modules. The default data source for the EJB modules is optional if data sources are specified for individual CMP beans.

Panel	Description
Map EJB references to beans	<p>On the Map EJB references to beans panel, if your application defines EJB references, you can specify JNDI names for enterprise beans that represent the logical names specified in EJB references.</p> <p>If the EJB reference is from EJB 3.0, Web 2.4, or Web 2.5 module, the JNDI name is optional. For earlier modules, each EJB reference defined in the application must be bound to an EJB file.</p> <p>If Allow EJB reference targets to resolve automatically is enabled, the JNDI name is optional for all modules. The product provides a container default value or automatically resolves the EJB reference for incomplete bindings.</p>
Map resource references to resources	<p>If your application defines resource references, for Map resource references to resources, specify JNDI names for the resources that represent the logical names defined in resource references. You can optionally specify login configuration name and authentication properties for the resource. After specifying authentication properties, click OK to save the values and return to the mapping step. You can optionally specify extended data source properties to enable a data source that uses heterogeneous pooling to connect to a DB2 database. Each resource reference defined in the application must be bound to a resource defined in your WebSphere Application Server configuration before clicking on Finish on the Summary panel.</p>
Map virtual hosts for Web modules	<p>If your application uses Web modules, for Map virtual hosts for Web modules, select a virtual host from the list that should map to a Web module defined in the application. The port number specified in the virtual host definition is used in the URL that is used to access artifacts such as servlets and JSP files in the Web module. Each Web module must have a virtual host to which it maps. Not specifying all needed virtual hosts will result in a validation error displaying after you click Finish on the Summary panel.</p>
Map security roles to users or groups	<p>If the application has security roles defined in its deployment descriptor then, for Map security roles to users or groups, specify users and groups that are mapped to each of the security roles. Select Role to select all of the roles or select individual roles. For each role, you can specify whether predefined users such as Everyone or All authenticated users are mapped to it. To select specific users or groups from the user registry:</p> <ol style="list-style-type: none"> 1. Select a role and click Lookup users or Lookup groups. 2. On the Lookup users or groups panel displayed, enter search criteria to extract a list of users or groups from the user registry. 3. Select individual users or groups from the results displayed. 4. Click OK to map the selected users or groups to the role selected on the Map security roles to users or groups panel.
Map RunAs roles to users	<p>If the application has Run As roles defined in its deployment descriptor, for Map RunAs roles to users, specify the Run As user name and password for every Run As role. Run As roles are used by enterprise beans that must run as a particular role while interacting with another enterprise bean. Select Role to select all of the roles or select individual roles. After selecting a role, enter values for the user name, password, and verify password and click Apply.</p>
Ensure all unprotected 1.x methods have the correct level of protection	<p>If your application contains EJB 1.x CMP beans that do not have method permissions defined for some of the EJB methods, for Ensure all unprotected 1.x methods have the correct level of protection, specify if you want to leave such methods unprotected or assign protection with deny all access.</p>
Bind listeners for message-driven beans	<p>If your application contains message driven enterprise beans, for Bind listeners for message-driven beans, provide a listener port name or an activation specification JNDI name for every message driven bean.</p>

Panel	Description
Map default data sources for modules containing 2.x entity beans	<p>If your application uses EJB modules that contain CMP beans that are based on the EJB 2.x specification, for Map default data sources for modules containing 2.x entity beans, specify a JNDI name for the default data source and the type of resource authorization to be used for the default data source for the EJB modules. You can optionally specify a login configuration name and authentication properties for the data source. When creating authentication properties, you must click OK to save the values and return to the mapping step. You can optionally specify extended data source properties to enable a data source that uses heterogeneous pooling to connect to a DB2 database. The default data source for EJB modules is optional if data sources are specified for individual CMP beans.</p>
Map data sources for all 2.x CMP beans	<p>If your application has CMP beans that are based on the EJB 2.x specification, on the Map data sources for all 2.x CMP beans panel, for each of the 2.x CMP beans specify a JNDI name and the type of resource authorization for data sources to be used.</p> <p>You can optionally specify a login configuration name and authentication properties for the data source. When creating authentication properties, you must click OK to save the values and return to the mapping step. The data source attribute is optional for individual CMP beans if a default data source is specified for the EJB module that contains CMP beans. If neither a default data source for the EJB module nor a data source for individual CMP beans are specified, then a validation error is displayed after you click Finish and installation is cancelled.</p>
Ensure all unprotected 2.x methods have the correct level of protection	<p>If your application contains EJB 2.x CMP beans that do not have method permissions defined in the deployment descriptors for some of the EJB methods, on the Ensure all unprotected 2.x methods have the correct level of protection panel, specify whether you want to assign a specific role to the unprotected methods, add the methods to the exclude list, or mark them as unchecked. Methods added to the exclude list are marked as uncallable. For methods marked unchecked no authorization check is performed prior to their invocation.</p>
Provide options to perform the EJB Deploy	<p>If the Deploy enterprise beans setting is enabled on the Select installation options panel, then you can specify options for the EJB deployment tool on the Provide options to perform the EJB Deploy panel. On this panel, you can specify extra class paths, RMIC options, database types, and database schema names to be used while running the EJB deployment tool.</p> <p>You can specify the EJB deployment tool options on this panel when installing or updating an application that contains EJB modules. The EJB deployment tool runs during installation of EJB 1.x or 2.x modules. The EJB deployment tool does not run during installation of EJB 3.0 modules.</p>
Map shared libraries	<p>On the Shared library references and Shared library mapping panels, specify shared library files for your application or Web modules to use. A defined shared library must exist to associate your application or module to the library file.</p>
Map shared library relationships	<p>On the Map shared library relationships panel, specify relationship identifiers and composition unit names for shared libraries that modules in your enterprise application reference.</p> <p>When installing your enterprise application, the product creates a composition unit for each shared library relationship in the business-level application that you specified for Business-level application name on the Select installation options panel.</p>
Provide JSP reloading options for Web modules	<p>If your application uses Web modules, for Provide JSP reloading options for Web modules, configure the class reloading of JavaServer Pages (JSP) files.</p>

Panel	Description
Map context roots for Web modules	<p>If your application uses Web modules that are defined in the application XML deployment descriptor, for Map context roots for Web modules, specify a context root for each Web module in the application.</p> <p>The product does not include Web modules from annotations on this panel.</p>
Initialize parameters for servlets	<p>If your application uses Web modules that support Servlet 2.5, for Initialize parameters for servlets, specify or override initial parameters that are passed to the init method of Web module servlet filters.</p> <p>This panel shows servlets from the module XML deployment descriptor. Servlet deployment information from annotations is not available on this panel.</p>
Map environment entries for Web modules	<p>If your application uses Web modules that support Servlet 2.5, for Map environment entries for Web modules, configure the environment entries of Web modules such as servlets and JSP files.</p> <p>This panel shows environment entries from the module XML deployment descriptor. Environment entry deployment information from annotations is not available in this panel.</p>
Map resource environment entry references to resources	<p>If your application contains resource environment references, for Map resource environment entry references to resources, specify JNDI names of resources that map to the logical names defined in resource environment references. If each resource environment reference does not have a resource associated with it, after you click Finish a validation error is displayed.</p>
Correct use of system identity	<p>If your application defines Run-As Identity as <i>System Identity</i>, for Correct use of system identity, you can optionally change it to <i>Run-As role</i> and specify a user name and password for the Run As role specified. Selecting <i>System Identity</i> implies that the invocation is done using the WebSphere Application Server security server ID and should be used with caution as this ID has more privileges.</p>
Correct isolation levels for all resource references	<p>If your application has resource references that map to resources that have an Oracle database doing backend processing, for Correct isolation levels for all resource references, specify or correct the isolation level to be used for such resources when used by the application. Oracle databases support ReadCommitted and Serializable isolation levels only.</p>
Bind message destination references to administered objects	<p>If your application uses message driven beans, for Bind message destination references to administered objects, specify the JNDI name of the J2C administered object to bind the message destination reference to the message driven beans.</p> <p>If the message destination reference is from a EJB 3.0 module, then the JNDI name is optional and the run time provides a container default value.</p> <p>Note: If multiple message destination references link to the same message destination, only one JNDI name is collected. When a message destination reference links to the same message destination as a message driven bean and the destination JNDI name has been collected already, the destination JNDI name for the message destination reference is not collected.</p>
Provide JNDI names for JCA objects	<p>If your application contains an embedded .rar file, for Provide JNDI names for JCA objects, specify the name and JNDI name of each J2C connection factory, J2C administered object and J2C activation specification.</p>
Bind J2C activationspecs to destination JNDI names	<p>If your application contains an embedded .rar file, its activationSpec property has the value <i>Destination</i>, and its introspected type is <i>javax.jms.Destination</i>, for Bind J2C activationspecs to destination JNDI names, specify the <i>jndiName</i> value for each activation bound to it.</p>

Panel	Description
Select current backend ID	<p>If your application has EJB modules for which deployment code has been generated for multiple backend databases using an assembly tool, for Select current backend ID, specify the backend ID representing the backend database to be used when the EJB module runs.</p> <p>This step is not shown if the Deploy enterprise beans setting is enabled on the Select installation options panel and if a database type other than None is specified on the Provide options to perform the EJB Deploy panel.</p>
Metadata for modules	<p>If your application has EJB 3.0 or Web 2.5 modules, you can lock deployment descriptors for one or more of the EJB 3.0 or Web 2.5 modules. If you set the <code>metadata-complete</code> attribute to <code>true</code> and lock deployment descriptors, the product writes the complete module deployment descriptor, including deployment information from annotations, to XML format.</p>
Provide options to perform the Web services deployment	<p>If the Deploy Web services setting is enabled on the Select installation options panel and your application uses Web services, then you can specify <code>wsdeploy</code> command options on the Provide options to perform the Web services deployment panel. For information on this panel, refer to descriptions of the <code>wsdeploy -cp</code> and <code>-jardir</code> options.</p>

5. On the Summary panel, verify the cell, node, and server onto which the application modules will install:
 - a. Beside **Cell/Node/Server**, click **Click here**.
 - b. Verify the settings.
 - c. Return to the Summary panel.
 - d. Click **Finish**.

Results

Several messages are displayed, indicating whether your application file is installing successfully.

If **Validate input off/warn/fail** on the **Select installation options** panel is set to **warn**, the default, several validation warnings might be displayed. If the setting is **fail**, the validation warnings might cause errors.

If you receive an `OutOfMemory` error and the source application file does not install, your system might not have enough memory or your application might have too many modules in it to install successfully onto the server. If lack of system memory is not the cause of the error, package your application again so the `.ear` file has fewer modules.

If lack of system memory and the number of modules are not the cause of the error, check the options you specified on the Java virtual machine page of the application server running the administrative console. You might increase the maximum heap size. Then, try installing the application file again.

What to do next

After the application file installs successfully, do the following:

1. Save the changes to your configuration.

The application is registered with the administrative configuration and application files are copied to the target directory, which is `app_server_root/installedApps/cell_name` by default or the directory that you designate.

For a single-server product, application files are copied to the destination directory when the changes are saved.

If you clicked the **Save** link in the application installation messages, the Preparing for the application installation panel displays again. Click **Applications** → **Application Types** → **WebSphere enterprise applications** to exit the panel and to see your application in the list of installed applications.

2. Start the application.
3. Test the application. For example, point a Web browser at the URL for the deployed application and examine the performance of the application. If necessary, edit the application configuration.

Preparing for application installation settings

Use this page to specify an application or module to install.

To view this administrative console page, click **Applications** → **New application** → **New Enterprise Application**.

This page is the first Preparing for the application installation page. On this page, specify an application or module to install. You can install an enterprise application archive (EAR file), enterprise bean (EJB) module (JAR file), Session Initiation Protocol (SIP) module (SAR file), or Web module (WAR file).

The second Preparing for the application installation page has more installation options, such as to generate default bindings for incomplete existing bindings in your application or module.

Path to the new application:

Specifies the fully qualified path to the enterprise application file.

The file can be an .ear, .jar, .sar, or .war file.

During application installation, the product typically uploads application files from a client workstation running the browser to the server running the administrative console, and then deploys the application files on the server. In such cases, use the Web browser running the administrative console to select EAR, WAR, SAR, or JAR modules to upload to the server.

Use **Local file system** when the browser and application files are on the same computer.

Use **Remote file system** in the following situations:

- The application file resides on any node in the current cell context. Only .ear, .jar, .sar, or .war files are shown during the browsing.
- The application file resides on the file system of any of the nodes in a cell.
- The application file already resides on the computer running the application server. For example, the field value might be *profile_root/installableApps/test.ear*.

After the product transfers the application file, the **Remote file system** value shows the path of the temporary location on the server.

Preparing for application installation binding settings

Use this page to select whether to view all installation options and to change the existing bindings for you application or module during installation. You can chose to generate default bindings for any incomplete bindings in the application or module or to assign specific bindings during installation.

This page is the second Preparing for the application installation page.

To view this administrative console page, click **Applications** → **New application** → **New Enterprise Application**, specify the path for the application or module to install, and then click **Next**.

The console page might not display all of the binding options listed in this topic. The contents of the application or module that you are installing determines which options are displayed on the console page. Also, the **Specify bindings to use** option displays only when updating an installed application.

How do you want to install the application?:

Specifies whether to show only installation options that require you to supply information or to show all installation options.

Option	Description
Fast Path - Prompt only when additional information is required	Displays only those options that require your attention, based on the contents of your application or module. Use the fast path to install your application more easily because you do not need to examine all available installation options.
Detailed - Show all installation options and parameters	Displays all available installation options.

Specify bindings to use:

Specifies whether to merge bindings when you update applications or to use new or existing bindings.

This setting is shown only when you update an installed application, and not when you install a new application.

Option	Description
Merge new and existing bindings	Keeps the existing binding values of the installed application and adds new binding values in the updated application for incomplete bindings. Use merge if your updated application has binding values that differ from values specified for the installed application. The product assigns binding values in the following order: <ol style="list-style-type: none"> 1. Use existing binding values in the installed application. 2. If the installed application does not have a binding value, use the new binding value. 3. If both the installed application and the updated application do not have a binding value, use the default value. The product assigns a default value only if you select Generate default bindings.
Use new bindings	Uses binding values in the updated application. Does not use existing binding values in the installed application.
Use existing bindings	Uses existing binding values in the installed application. Does not use binding values in the updated application.

Generate default bindings:

Specifies whether to generate default bindings and mappings. To view this setting, expand **Choose to generate default bindings and mappings**. If you select **Generate default bindings**, then the product completes any incomplete bindings in the application with default values. The product does not change existing bindings.

After you select **Generate default bindings**, you can advance directly to the Summary step and install the application if none of the steps have a red asterisk (*). A red asterisk denotes that the step has incomplete data and requires a valid value. On the Summary panel, verify the cell, node, and server on which the application is installed.

Note: You do not need to specify Java Naming and Directory Interface (JNDI) values for EJB bean, local home, remote home, or business interfaces of EJB 3.0 modules. The product assigns container default values during run time. Similarly, for any EJB reference within an EJB 3.0, Web 2.4, or Web 2.5 module, you do not need to specify JNDI values because the product resolves the targets automatically during run time. Even when you select **Generate default bindings**, the product does not generate default values for those JNDI values but it does generate default values for other bindings such as virtual host.

If you select **Generate default bindings**, the product generates bindings as follows:

- Enterprise bean (EJB) JNDI names are generated in the form *prefix/ejb-name*. The default prefix is *ejb*, but can be overridden. The *ejb-name* is as specified in the deployment descriptors `<ejb-name>` tag or in its corresponding annotation for EJB 3.0 modules. The product does not generate default values for enterprise beans in an EJB 3.0 module because the run time provides container default values.
- EJB references are bound if an `<ejb-link>` is found. Otherwise, if a unique enterprise bean is found with a matching home (or local home) interface as the referenced bean, the reference is resolved automatically. The product does not generate default values for EJB reference in an EJB 3.0, Web 2.4, or Web 2.5 module because the run time provides container default values or automatically resolves the target references.
- Resource reference bindings are derived from the `<res-ref-name>` tag or its corresponding annotation for Java Platform, Enterprise Edition (Java EE) 5 modules. This action assumes that the `java:comp/env` name is the same as the resource global JNDI name.
- Connection factory bindings for EJB 2.0 and EJB 2.1 JAR files are generated based on the JNDI name and authorization information provided. This action results in default connection factory settings for each EJB 2.0 and EJB 2.1 JAR file in the application being installed. No bean-level connection factory bindings are generated.
- Data source bindings for EJB 1.1 JAR files are generated based on the JNDI name, data source user name password options. This action results in default data source settings for each JAR file. No bean-level data source bindings are generated.
- For EJB 2.0 or later message-driven beans deployed as Java EE Connector Architecture (JCA) 1.5-compliant resources, the JNDI names corresponding to `activationSpec` instances are generated in the form `eis/MDB_ejb-name`. Message destination references are bound if a `<message-destination-link>` is found, then the JNDI name is set to `ejs/message-destination-linkName`. Otherwise, the JNDI name is set to `eis/message-destination-refName`.
- For EJB 2.0 or later message-driven beans deployed against listener ports, the listener ports are derived from the message-driven bean `<ejb-name>` tag with the string `Port` appended.
- For `.war` files, the virtual host is set as `default_host` unless otherwise specified.

The default strategy suffices for most applications or at least for most bindings in most applications. However, if you experience errors, complete the following actions:

- Control the global JNDI names of one or more EJB files.
- Control data source bindings for container-managed persistence (CMP) beans. That is, you have multiple data sources and need more than one global data source.
- Map resource references to global resource JNDI names that are different from the `java:comp/env` name.

In such cases, you can change the behavior with an XML document, which is a custom strategy. Use the **Specific bindings file** setting to specify a custom strategy and see the setting description in this help file for examples.

Override existing bindings:

Specifies whether generated bindings are to replace existing bindings.

The default is to not override existing bindings. Select **Override existing bindings** to have generated bindings replace existing bindings.

Specific bindings file:

Specifies a bindings file that overrides the default binding.

Change the behavior of the default binding with an XML document, which is a custom strategy. Custom strategies extend the default strategy so you only need to customize those areas where the default strategy is insufficient. Thus, you only need to describe how you want to change the bindings generated by the default strategy; you do not have to define bindings for the entire application.

Use the following examples to override various aspects of the default bindings generator:

Controlling an EJB JNDI name

```
<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>helloEjb.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>HelloEjb</ejb-name>
          <jndi-name>com/acme/ejb/HelloHome</jndi-name>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

Note: Ensure that the setting for <ejb-name> matches the ejb-name entry in the EJB JAR deployment descriptor. Here the setting is <ejb-name>HelloEjb</ejb-name>.

Setting the connection factory binding for an EJB JAR file

```
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>yourEjb20.jar</jar-name>
      <connection-factory>
        <jndi-name>eis/jdbc/YourData_CMP</jndi-name>
        <res-auth>Container</res-auth>
      </connection-factory>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

Setting the connection factory binding for an EJB file

```
<?xml version="1.0">
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>yourEjb20.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>YourCmp20</ejb-name>
          <connection-factory>
            <jndi-name>eis/jdbc/YourData_CMP</jndi-name>
            <res-auth>PerConnFact</res-auth>
          </connection-factory>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

Note: Ensure that the setting for <ejb-name> matches the ejb-name tag in the deployment descriptor. Here the setting is <ejb-name>YourCmp20</ejb-name>.

Setting the message destination reference JNDI for a specific enterprise bean

This example shows an XML extract in a custom strategy file for setting message-destination-refs for a specific enterprise bean.

```
<?xml version="1.0">
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>yourEjb21.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>YourSession21</ejb-name>
          <message-destination-ref-bindings>
            <message-destination-ref-binding>
              <message-destination-ref-name>jdbc/MyDataSrc</message-destination-ref-name>
              <jndi-name>eis/somA0</jndi-name>
            </message-destination-ref-binding>
          </message-destination-ref-bindings>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

Note: Ensure that the setting for `<ejb-name>` matches the `ejb-name` tag in the deployment descriptor. Here the setting is `<ejb-name>YourSession21</ejb-name>`. Also ensure that the setting for `<message-destination-ref-name>` matches the `message-destination-ref-name` tag in the deployment descriptor. Here the setting is `<message-destination-ref-name>jdbc/MyDataSrc</message-destination-ref-name>`.

Overriding a resource reference binding from a WAR, EJB JAR file, or Java EE client JAR file

This example shows code for overriding a resource reference binding from a WAR file. Use similar code to override a resource reference binding from an enterprise bean (EJB) JAR file or a Java EE client JAR file.

```
<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <war-binding>
      <jar-name>hello.war</jar-name>
      <resource-ref-bindings>
        <resource-ref-binding>
          <resource-ref-name>jdbc/MyDataSrc</resource-ref-name>
          <jndi-name>war/override/dataSource</jndi-name>
        </resource-ref-binding>
      </resource-ref-bindings>
    </war-binding>
  </module-bindings>
</dfltbndngs>
```

Note: Ensure that the setting for `<resource-ref-name>` matches the `resource-ref` tag in the deployment descriptor. In the previous example, the setting is `<resource-ref-name>jdbc/MyDataSrc</resource-ref-name>`.

Overriding the JNDI name for a message-driven bean deployed as a JCA 1.5-compliant resource

This example shows an XML extract in a custom strategy file for overriding the Java Message Service (JMS) activationSpec JNDI name for an EJB 2.0 or later message-driven bean deployed as a JCA 1.5-compliant resource.

```
<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
```

```

<module-bindings>
  <ejb-jar-binding>
    <jar-name>YourEjbJar.jar</jar-name>
    <ejb-bindings>
      <ejb-binding>
        <ejb-name>YourMDB</ejb-name>
        <activation-spec-jndi-name>activationSpecJNDI</activation-spec-jndi-name>
      </ejb-binding>
    </ejb-bindings>
  </ejb-jar-binding>
</module-bindings>
</dfltbindings>

```

Overriding the JMS listener port name for an EJB 2.0, 2.1, or 3.0 message-driven bean

This example shows an XML extract in a custom strategy file for overriding the JMS listener port name for an EJB 2.0 or later message-driven bean deployed against a listener port.

```

<?xml version="1.0"?>
<!DOCTYPE dfltbindings SYSTEM "dfltbindings.dtd">
<dfltbindings>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>YourEjbJar.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>YourMDB</ejb-name>
          <listener-port>yourMdbListPort</listener-port>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbindings>

```

Overriding an EJB reference binding from an EJB JAR, WAR file, or EJB file

This example shows code for overriding an EJB reference binding from an EJB JAR file. Use similar code to override an EJB reference binding from a WAR file or an EJB file.

```

<?xml version="1.0"?>
<!DOCTYPE dfltbindings SYSTEM "dfltbindings.dtd">
<dfltbindings>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>YourEjbJar.jar</jar-name>
      <ejb-ref-bindings>
        <ejb-ref-binding>
          <ejb-ref-name>YourEjb</ejb-ref-name>
          <jndi-name>YourEjb/JNDI</jndi-name>
        </ejb-ref-binding>
      </ejb-ref-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbindings>

```

Specify unique prefix for beans:

Specifies a string that the product applies to the beginning of generated enterprise bean JNDI names. The prefix must be unique within the cell or node.

The default is to not specify a unique prefix for beans.

Default bindings for EJB 1.1 CMP beans:

Specifies the default data source JNDI name and other bindings for container-managed persistence (CMP) 1.1 beans.

The default is to not use default bindings for EJB 1.1 CMP beans.

If you select **Default bindings for EJB 1.1 CMP beans**, specify the JNDI name for the default data source to be used with the CMP 1.1 beans. Also specify the user name and password for this default data source.

Default connection factory bindings:

Specifies the default connection factory JNDI name.

The default is to not use default connection factory bindings. Select **Default connection factory bindings** to specify bindings for connection factories.

If you select **Default connection factory bindings**, specify the JNDI name for the default connection factory to be used. Also specify whether the resource authorization is for the application or container-wide.

Use default virtual host name for Web and SIP modules:

Specifies the virtual host for the Web module (WAR file) or Session Initiation Protocol (SIP) module (SAR file).

The default is to not use default virtual host name for Web or SIP modules. If you select **Use default virtual host name for Web and SIP modules**, specify a default host name.

Select installation options settings

Use this panel to specify options for the installation of a Java Platform, Enterprise Edition (Java EE) application onto a WebSphere Application Server deployment target. Default values for the options are used if you do not specify a value. After application installation, you can specify values for many of these options from an enterprise application settings page.

To view this administrative console panel, click **Applications** → **New application** → **New Enterprise Application** and then specify values as needed for your application on the Preparing for application installation pages.

The Select installation options panel is the same for the application installation and update wizards.

Precompile JavaServer Pages files:

Specify whether to precompile JavaServer Pages (JSP) files as a part of installation. The default is not to precompile JSP files.

For this option, install only onto a Version 6.1 or later deployment target.

If you select **Precompile JavaServer Pages files** and try installing your application onto an earlier deployment target such as Version 5.x, the installation is rejected. You can deploy applications to only those deployment targets that have same version as the product. If applications are targeted to servers that have an earlier version than the product, then you cannot deploy to those targets.

Data type	Boolean
Default	false

Directory to install application:

Specifies the directory to which the enterprise archive (EAR) file will be installed.

By default, the EAR file is installed in the *profile_root/installedApps/cell_name/application_name.ear* directory.

Setting options include the following:

- Do not specify a value and leave the field empty.

The default value is `${APP_INSTALL_ROOT}/cell_name`, where the `${APP_INSTALL_ROOT}` variable is *profile_root/installedApps*. A directory having the EAR file name of the application being installed is appended to `${APP_INSTALL_ROOT}/cell_name`. Thus, if you do not specify a directory, the EAR file is installed in the *profile_root/installedApps/cell_name/application_name.ear* directory.

- Specify a directory.

If you specify a directory for **Directory to install application**, the application is installed in *specified_path/application_name.ear* directory. A directory having the EAR file name of the application being installed is appended to the path that you specify for **Directory to install application**. For example, if you are installing *Clock.ear* and specify `C:/myapps` on Windows machines, the application is installed in the *myapps/Clock.ear* directory. The `${APP_INSTALL_ROOT}` variable is set to the specified path.

- Specify `${APP_INSTALL_ROOT}/${CELL}` for the initial installation of the application.

If you intend to export the application from one cell and later install the exported application on a different cell, specify the `${CELL}` variable for the initial installation of the application. For example, specify `${APP_INSTALL_ROOT}/${CELL}` for this setting. Exporting the application creates an enhanced EAR file that has the application and its deployment configuration. The deployment configuration retains the cell name of the initial installation in the destination directory unless you specify the `${CELL}` variable. Specifying the `${CELL}` variable ensures that the destination directory has the current cell name, and not the original cell name.

Note: If an installation directory is not specified when an application is installed on a single-server configuration, the application is installed in `${APP_INSTALL_ROOT}/cell_name`. When the server is made a part of a multiple-server configuration (using the `addNode` utility), the cell name of the new configuration becomes the cell name of the deployment manager node. If the `-includeapps` option is used for the `addNode` utility, then the applications that are installed prior to the `addNode` operation still use the installation directory `${APP_INSTALL_ROOT}/cell_name`. However, an application that is installed after the server is added to the network configuration uses the default installation directory `${APP_INSTALL_ROOT}/network_cell_name`. To move the application to the `${APP_INSTALL_ROOT}/network_cell_name` location upon running the `addNode` operation, explicitly specify the installation directory as `${APP_INSTALL_ROOT}/${CELL}` during installation. In such a case, the application files can always be found under `${APP_INSTALL_ROOT}/current_cell_name`.

- If the application has been exported and you are installing the exported EAR file in a different cell or location, specify `${APP_INSTALL_ROOT}/cell_name/application_name.ear` if you did not specify `${APP_INSTALL_ROOT}/${CELL}` for the initial installation.

The exported EAR file is an enhanced EAR file that has the application and its deployment configuration. The deployment configuration retains the value for **Directory to install application** that was used for the previous installation of the application. Unless you specify a different value for **Directory to install application** for this installation, the enhanced EAR file will be installed to the same directory as for the previous installation.

If you did not specify the `${CELL}` variable during the initial installation, the deployment configuration uses the cell name of the initial installation in the destination directory. If you are installing on a different cell, specify `${APP_INSTALL_ROOT}/cell_name/application_name.ear`, where *cell_name* is the name of the cell to which you want to install the enhanced EAR file. If you do not designate the current cell name, *cell_name* will be the original cell name even though you are installing the enhanced EAR file on a cell that has a different name.

- Specify an absolute path or a use pathmap variable.
You can specify an absolute path or use a pathmap variable such as `${MY_APPS}`. You can use a pathmap variable in any installation.

This **Directory to install application** field is the same as the **Location (full path)** setting on an Application binaries page.

Data type	String
Units	Full path name

Distribute application:

Specifies whether the product expands application binaries in the installation location during installation and deletes application binaries during uninstallation. The default is to enable application distribution. Application binaries for installed applications are expanded to the directory specified.

On single-server products, the binaries are deleted when you uninstall and save changes to the configuration.

On multiple-server products, the binaries are deleted when you uninstall and save changes to the configuration and synchronize changes.

If you disable this option, then you must ensure that the application binaries are expanded appropriately in the destination directories of all nodes where the application runs.

Note: If you disable this option and you do not copy and expand the application binaries to the nodes, a later saving of the configuration or manual synchronization does not move the application binaries to the nodes for you.

This **Distribute application** field is the same as the **Enable binary distribution, expansion and cleanup post uninstallation** setting on an Application binaries page.

Data type	Boolean
Default	true

Use binary configuration:

Specifies whether the application server uses the binding, extensions, and deployment descriptors located with the application deployment document, the deployment.xml file (default), or those located in the enterprise archive (EAR) file. Select this setting for applications installed on Version 6.0 or later deployment targets only. This setting is not valid for applications installed on 5.x deployment targets.

The default (false) is not to use the binding, extensions, and deployment descriptors located in deployment.xml. To use the binding, extensions, and deployment descriptors located in the EAR file, enable this setting (true).

This **Use binary configuration** field is the same as the **Use configuration information in binary** setting on an Application binaries page.

Data type	Boolean
Default	false

Deploy enterprise beans:

Specifies whether the EJBDeploy tool runs during application installation.

The tool generates code needed to run enterprise bean (EJB) files. You must enable this setting in the following situations:

- The EAR file was assembled using an assembly tool such as Rational Application Developer and the EJBDeploy tool was not run during assembly.
- The EAR file was not assembled using an assembly tool such as Rational Application Developer.
- The EAR file was assembled using versions of the Application Assembly Tool (AAT) previous to Version 5.0.

The EJB deployment tool runs during installation of EJB 1.x or 2.x modules. The EJB deployment tool does not run during installation of EJB 3.0 modules.

For this option, install only onto a Version 6.1 or later deployment target.

If you select **Deploy enterprise beans** and try installing your application onto an earlier deployment target such as Version 6.0, the installation is rejected. You can deploy applications to only those targets that have same WebSphere version as the product. If applications are targeted to servers that have an earlier version than the product, then you cannot deploy to those targets.

Also, if you select **Deploy enterprise beans** and specify a database type on the **Provide options to perform the EJB Deploy** panel, previously defined backend IDs for all of the EJB modules are overwritten by the chosen database type. To enable backend IDs for individual EJB modules, set the database type to "" (null) on the **Provide options to perform the EJB Deploy** panel.

Enabling this setting might cause the installation program to run for several minutes.

Data type Boolean
Default true (false for EJB 3.0 modules)

Application name:

Specifies a logical name for the application. An application name must be unique within a cell and cannot contain an unsupported character.

An application name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

This **Application name** field is the same as the **Name** setting on an Enterprise application settings page.

Data type String

Create MBeans for resources:

Specifies whether to create MBeans for resources such as servlets or JSP files within an application when the application starts. The default is to create MBeans.

This field is the same as the **Create MBeans for resources** setting on a Startup behavior page.

Data type	Boolean
Default	true

Override class reloading settings for Web and EJB modules:

Specifies whether the product run time detects changes to application classes when the application is running. If this setting is enabled and if application classes are changed, then the application is stopped and restarted to reload updated classes.

The default is not to enable class reloading.

This field is the same as the **Override class reloading settings for Web and EJB modules** setting on an Class loading and update detection page.

Data type	Boolean
Default	false

Reload interval in seconds:

Specifies the number of seconds to scan the application's file system for updated files. The default is the value of the reloading interval attribute in the IBM extension (META-INF/ibm-application-ext.xml) file of the EAR file.

The reloading interval attribute takes effect only if class reloading is enabled.

To enable reloading, specify a value greater than zero (for example, 1 to 2147483647). To disable reloading, specify zero (0). The range is from 0 to 2147483647.

This **Reload interval in seconds** field is the same as the **Polling interval for updated files** setting on a Class loading and update detection page.

Data type	Integer
Units	Seconds
Default	3

Deploy Web services:

Specifies whether the Web services deploy tool wsdeploy runs during application installation.

The tool generates code needed to run applications using Web services. The default is not to run the wsdeploy tool. You must enable this setting if the EAR file contains modules using Web services and has not previously had the wsdeploy tool run on it, either from the **Deploy** menu choice of an assembly tool or from a command line.

For this option, install only onto a Version 6.1 or later deployment target.

If you select **Deploy Web services** and try installing your application onto an earlier deployment target such as Version 5.x, the installation is rejected. You can deploy applications to only those targets that have same version as the product. If applications are targeted to servers that have an earlier version than the

product, then you cannot deploy to those targets.

Data type Boolean
Default false

Validate input off/warn/fail:

Specifies whether the product examines the application references specified during application installation or updating and, if validation is enabled, warns you of incorrect references or fails the operation.

An application typically refers to resources using data sources for container managed persistence (CMP) beans or using resource references or resource environment references defined in deployment descriptors. The validation checks whether the resource referred to by the application is defined in the scope of the deployment target of that application.

Select **off** for no resource validation, **warn** for warning messages about incorrect resource references, or **fail** to stop operations that fail as a result of incorrect resource references.

This **Validate input off/warn/fail** field is the same as the **Application reference validation** setting on an Enterprise application settings page.

Data type String
Default warn

Process embedded configuration:

Specifies whether the embedded configuration should be processed. An embedded configuration consists of files such as resource.xml and variables.xml. When selected or true, the embedded configuration is loaded to the application scope from the .ear file. If the .ear file does not contain an embedded configuration, the default is false. If the .ear file contains an embedded configuration, the default is true.

Data type Boolean
Default false

File permission:

Specifies access permissions for application binaries for installed applications that are expanded to the directory specified.

The **Distribute application** option must be enabled to specify file permissions.

You can specify file permissions in the text field. You can also set some of the commonly used file permissions by selecting them from the multiple-selection list. List selections overwrite file permissions set in the text field.

You can set one or more of the following file permission strings in the list. Selecting multiple options combines the file permission strings.

Multiple-selection list option	File permission string set
Allow all files to be read but not written to	.*=755
Allow executables to execute	.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
Allow HTML and image files to be read by everyone	.*\.htm=755#.*\.html=755#.*\.gif=755#.*\.jpg=755

Instead of using the multiple-selection list to specify file permissions, you can specify a file permission string in the text field. File permissions use a string that has the following format:

file_name_pattern=permission#file_name_pattern=permission

where *file_name_pattern* is a regular expression file name filter (for example, *.*\\.jsp* for all JSP files), *permission* provides the file access control lists (ACLs), and *#* is the separator between multiple entries of *file_name_pattern* and *permission*. If *#* is a character in a *file_name_pattern* string, use *\\#* instead.

If multiple file name patterns and file permissions in the string match a uniform resource identifier (URI) within the application, then the product uses the most stringent applicable file permission for the file. For example, if the file permission string is *.*\\.jsp=775#a.*\\.jsp=754*, then the *abc.jsp* file has file permission 754.

Note: Using regular expressions for file matching pattern compares an entire string URI against the specified file permission pattern. You must provide more precise matching patterns using regular expressions as defined by Java programming API. For example, suppose the following directory and file URIs are processed during a file permission operation:

1	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war
2	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp
3	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/META-INF/MANIFEST.MF
4	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/WEB-INF/classes/MyClass.class
5	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/mydir/MyClass2.class
6	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/META-INF

The file pattern matching results are:

- *MyWarModule.war* does not match any of the URIs
- *.*MyWarModule.war.** matches all URIs
- *.*MyWarModule.war\$* matches only URI 1
- *.*\\.jsp=755* matches only URI 2
- *.*META-INF.** matches URIs 3 and 6
- *.*MyWarModule.war/.*/.*\\.class* matches URIs 4 and 5

If you specify a directory name pattern for **File permissions**, then the directory permission is set based on the value specified. Otherwise, the **File permissions** value set on the directory is the same as its parent. For example, suppose you have the following file and directory structure:

/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp

and you specify the following file pattern string:

*.*MyApp.ear\$=755#.*\\.jsp=644*

The file pattern matching results are:

- Directory *MyApp.ear* is set to 755
- Directory *MyWarModule.war* is set to 755
- Directory *MyWarModule.war* is set to 755

Note: Regardless of the operation system, always use a forward slash (/) as a file path separator in file patterns.

Access permissions specified here are at the application level. You can also specify access permissions for application binaries in the node-level configuration. The node-level file permissions specify the maximum (most lenient) permissions that can be given to application binaries. Access permissions specified here at application level can only be the same as or more restrictive than those specified at the node level.

This setting is the same as the **File permissions** field on the Application binaries page.

Data type String

Application build identifier:

Specifies an uneditable string that identifies the build version of the application.

This **Application build identifier** field is the same as the **Application build level** field on the Application binaries page.

Data type String

Business-level application name:

Specifies whether the product creates a new business-level application with the enterprise application that you are installing or makes the enterprise application a composition unit of an existing business-level application.

The default is to create a new business-level application with a setting value of `WebSphere:blaname=Anyasset,blaedition=BASE`. When you select to create a new business-level application from the drop-down list, the product creates a business-level application that has the same name as your enterprise application.

To add your enterprise application to an existing business-level application, select an existing business-level application from the drop-down list. The product makes your enterprise application a composition unit of the existing business-level application.

Data type String
Default Create a new business-level application that has the same name as the enterprise application that you are installing.

`WebSphere:blaname=Anyasset,blaedition=BASE`

Asynchronous request dispatch type:

Specifies whether Web modules can dispatch requests concurrently on separate threads and, if so, whether the server or client dispatches the requests. Concurrent dispatching can improve servlet response time.

If operations are dependant on each other, do not enable asynchronous request dispatching. Select **Disabled**. Concurrent dispatching might result in errors when operations are dependant.

Select **Server side** to enable the server to dispatch requests concurrently. Select **Client side** to enable the client to dispatch requests concurrently.

Data type String
Default Disabled

Allow EJB reference targets to resolve automatically:

Specifies whether the product assigns default JNDI values for or automatically resolves incomplete EJB reference targets.

Select this option to enable EJB reference targets to resolve automatically if the references are from EJB 2.1 or earlier modules or from Web 2.3 or earlier modules. If you enable this option, the runtime container provides a default value or automatically resolves the EJB reference for any EJB reference that does not have a binding.

If you selected **Generate default bindings** on the Preparing for application installation page, then you do not need to select this option. The product generates default values.

If you select **Allow EJB reference targets to resolve automatically**, all modules in the application must share one deployment target. If you select this option and all of the application modules do not share a common server, after you click **Finish** on the Summary page, the product displays a warning message and does not install the application. You must deselect this setting before you click **Finish** to install the application.

Data type	Boolean
Default	false

Provide options to perform the EJB Deploy settings

Use this panel to specify options for the enterprise bean (EJB) deployment tool. The tool generates code needed to run enterprise bean files. You can specify extra class paths, Remote Method Invocation compiler (RMIC) options, database types, and database schema names to be used while running the EJB deployment tool.

This administrative console panel is a step in the application installation and update wizards. To view this panel, you must select **Deploy enterprise beans** on the **Select installation options** panel. Thus, to view this panel, click **Applications** → **New Application** → **New Enterprise Application** → *application_path* → **Next** → **Detailed - Show all installation options and parameters** → **Next** → **Deploy enterprise beans** → **Next** → **Step: Provide options to perform the EJB Deploy**.

You can specify the EJB deployment tool options on this panel when installing or updating an application that contains EJB modules. The EJB deployment tool runs during installation of EJB 1.x or 2.x modules. The EJB deployment tool does not run during installation of EJB 3.0 modules.

The options that you specify set parameter values for the `ejbdeploy` command. The tool, and thus the `ejbdeploy` command, is run on the enterprise archive (EAR) file during installation after you click **Finish** on the **Summary** panel of the wizard.

Class path:

Specifies the class path of one or more zipped or Java archive (JAR) files on which the JAR or EAR file being installed depends.

To specify the class paths of multiple zipped and JAR files, the zipped and JAR file names must be fully qualified, separated by semicolons, and enclosed in double quotation marks. For example:

```
path\myJar1.jar;path\myJar2.jar;path\myJar3.jar
```

Class path is the same as the `ejbdeploy` command parameter `-cp class_path`.

Data type	String
Default	null

RMIC:

Specifies whether the EJB deployment tool passes RMIC options to the Remote Method Invocation compiler. Refer to RMI Tools documentation for information on the options.

Separate options by a space and enclose them in double quotation marks. For example:

```
"-nowarn -verbose"
```

The **RMIC** setting is the same as the ejbdeploy command parameter `-rmic "options"`.

Data type	String
Default	null

Database type:

Specifies the name of the database vendor, which is used to determine database column types, mapping information, Table.sql, and other information. Select a database type or the empty choice from the drop-down list. The list contains the names of valid database vendors. Selecting the empty choice sets the database type to "" (null).

If you specify a database type, previously defined backend IDs for all of the EJB modules are overwritten by the chosen database type. To enable backend IDs for individual EJB modules, select the empty choice to set the database type to null.

The backend IDs SQL92 (1992 SQL Standard) and SQL99 (1999 SQL Standard) are deprecated. Although the SQL92 and SQL99 backend IDs are available in the list, they are deprecated.

Database type is the same as the ejbdeploy command parameter `-dbvendor name`.

Data type	String
Default	DB2UDB_V82

Database schema:

Specifies the name of the schema that you want to create.

The EJB deployment tool saves database information in the schema document in the JAR or EAR file, which means that the options do not need to be specified again. It also means that when a JAR or EAR is generated, the correct database must be defined at that point because it cannot be changed later.

If the name of the schema contains any spaces, the entire name must be enclosed in double quotes. For example:

```
"my schema"
```

Database schema is the same as the ejbdeploy command parameter `-dbschema "name"`.

Data type	String
Default	null

Database access type:

Specifies the database access type for a DB2 database that supports Structured Query Language for Java (SQLJ). Use SQLJ to develop data access applications that connect to DB2 databases. SQLJ is a set of

programming extensions that support use of the Java programming language to embed statements that provide SQL (Structured Query Language) database requests.

To view this setting, you must select a DB2 backend database that supports SQLJ from the **Database type** drop-down list.

Available database access types include JDBC and SQLJ.

Data type	String
Default	JDBC

SQLJ class path:

Specifies the class path of the DB2 SQLJ tool sqlj.zip file. The product uses this class path to run the DB2 SQLJ tool during application installation and generate SQLJ profiles (.ser files).

To view this setting, you must select a DB2 backend database that supports SQLJ from the **Database type** drop-down list.

When you reinstall an application EAR file, the product deletes any existing SQLJ profiles and creates new profiles.

If you do not specify a class path, the product displays a warning about the missing class path. After you specify a valid class path, you can continue using the wizard for the application installation.

You can customize or add bindings to the generated SQLJ profile after the product installs the application. Use the administrative console SQLJ profiles and pureQuery bind files page accessed by clicking **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **SQLJ profiles and pureQuery bind files**.

Data type	String
Default	null

JDK compliance level:

Specifies the Java developer kit compiler compliance level as *1.4*, *5.0*, or *6.0* when you include application source files for compilation.

The default is to use whatever developer kit version the ejbdeploy command is using. If your application is using new functionality defined in Version 5.0 or 6.0 or you are including source files (which is not recommended), then you must specify the Version 5.0 or 6.0 level.

JDK compliance level is the same as the ejbdeploy command parameter `-complianceLevel "1.4" | "5.0" | "6.0"`.

Data type	String
Default	null (empty string)

Bind listeners for message-driven beans settings

Use this panel to specify bindings for message-driven beans in your application or module.

To view this administrative console panel, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Message Driven Bean listener bindings**. This panel is the same as the **Bind listeners for message-driven beans** panel on the application installation and update wizards.

Each message-driven bean must be bound to a listener port name or to an activation specification Java Naming and Directory Interface (JNDI) name.

Provide a listener port name if your application uses any of the following Java Message Service (JMS) providers:

- Version 5 default messaging
- WebSphere MQ
- Generic

Provide an activation specification JNDI name if your application's resources are configured using the default messaging provider or any generic J2C resource adapter that supports inbound messaging.

Not providing valid listener port names or activation specification JNDI names results in the following errors:

- If neither a listener port name or an activation specification JNDI name is specified for a message driven bean, then a validation error is displayed after you click **Finish** on the **Summary** panel.
- If the module containing the message-driven bean is deployed on a 5.x deployment target and a listener port is not specified, then a validation error is displayed after you click **Next**.
- If multiple message driven beans are linked to the same destination, specify the same destination JNDI name for each message driven bean. If you specify different destination JNDI names, a validation error is displayed and all JNDI specifications after the first one are ignored.

To apply binding changes to multiple mappings:

1. In the list of mappings, select the **Select** check box beside each EJB module that you want mapped to a particular binding.
2. Expand **Apply Multiple Mappings**.
3. Specify a listener port name or select a target resource JNDI name for an activation specification.
4. If you are defining a binding for an activation specification, optionally specify the following:

Destination JNDI name

For resource adapters that support JMS, specify `javax.jms.Destinations` so the resource adapter can service messages from the JMS destination. A destination JNDI name set as part of application deployment take precedence over properties set on an activation specification administrative object.

Target resource JNDI Name

Specify the target resource JNDI name when mapping a message-driven bean to an activation specification.

ActivationSpec authentication alias

Specify an authentication alias that is used to access the user name and password that are set on the configured J2C activation specification. Authentication alias properties set as part of application deployment take precedence over properties set on an activation specification administrative object.

5. Click **Apply**.
6. Click **OK** or **Next**.

EJB module:

Specifies the name of the module that contains the enterprise bean.

EJB:

Specifies name of an enterprise bean in the application.

URI:

Specifies the location of the module relative to the root of the application EAR file.

Messaging type:

Specifies the type of message-driven bean.

Bindings:

Specifies a listener port name or an activation specification JNDI name for the message-driven bean. When a message-driven enterprise bean is bound to an activation specification JNDI name you can also specify the destination JNDI name and the authentication alias.

Bindings specify JNDI names for the referenceable and referenced artifacts in an application. An example JNDI name for a listener port to be used by a Store application might be StoreMdbListener. The binding definition is stored in IBM bindings files such as ibm-ejb-jar-bnd.xmi.

Example: Installing an EAR file using the default bindings

If application bindings were not specified for all enterprise beans or resources in an enterprise application during application development or assembly, you can select to generate default bindings. After application installation, you can modify the bindings as needed using the administrative console.

Before you begin

This topic assumes that the application can run on a Web server.

About this task

This topic describes how to install a simple .ear file using the default bindings. You can follow the steps to install any application, including applications provided with the product in the samples or installableApps subdirectory.

1. Click **Applications** → **New application** → **New Enterprise Application** in the console navigation tree.
2. On the first Preparing for application install page, specify the full path name of the EAR file.
 - a. For **Path to the new application**, specify the full path name of the .ear file. For this example, the base file name is my_appl.ear and the file resides on a server at C:\sample_apps.
For this example, the base file name is my_appl.ear and the file resides on a server at /home/myuserid/myapps. Thus, enter the fully qualified path name for the file, /home/myuserid/myapps/my_appl.ear.
 - b. Click **Next**.
3. On the second Preparing for application install page, choose to generate default bindings.
 - a. Expand **Choose to generate default bindings and mappings**.
 - b. Select **Generate default bindings**.
Using the default bindings causes any incomplete bindings in the application to be filled in with default values. the product does not change existing bindings. By choosing this option, you can skip many of the steps of the application installation wizard and go directly to the Summary step.
 - c. Click **Next**.
4. If application security warnings are displayed, read the warnings and click **Continue**.
5. On the Install New Application page, click the step number for **Map modules to servers**, and verify the cell, node, and server onto which the application files will install.

- a. From the **Clusters and servers** list, select the server onto which the application files will install.
- b. Select all of the application modules.
- c. Click **Next**.

On the **Map modules to servers** panel, you can map modules to other servers such as Web servers. If you want a Web server to serve the application, use the **Ctrl** key to select an application server or cluster and the Web server together in order to have the plug-in configuration file `plugin-cfg.xml` for that Web server generated based on the applications which are routed through it.

6. On the Install New Application page, click the step number beside **Summary**, the last step.
7. On the Summary panel, click **Finish**.

What to do next

Examine the application installation progress messages. If the application installs successfully, save your administrative configuration. You can now see the name of your application in the list of deployed applications on the Enterprise applications page accessed by clicking **Applications** → **Application Types** → **WebSphere enterprise applications** in the console navigation tree.

If the application does not install successfully, read the messages to identify why the installation failed. Correct problems with the application as needed and try installing the application again.

If the application has a Web module, try opening a browser on the application.

1. Point a Web browser at the URL for the deployed application.
The URL typically has the format `http://host_name:9060/Web_module_name`, where *host_name* is your valid Web server and 9060 is the default port number.
2. Examine the performance of the application.

If the application does not perform as desired, edit the application configuration, then save and test it again.

Example: Installing a Web Services Sample with the console

The product provides a Web Services sample application that you can install on a Version 7.x application server.

Before you begin

During installation, select to install the sample applications. After installation, ensure that your product installation has a Version 7.x application server onto which you can install the Web Services Sample.

About this task

Installing the sample applications adds the `JaxWSServicesSamples.ear` enterprise application and supporting Java archives (JAR files) to the `app_server_root/samples/lib/JaxWSServicesSamples` directory of your product installation.

This topic describes how to install and start the `JaxWSServicesSamples.ear` enterprise application using an administrative console.

1. Click **Applications** → **New application** → **New Enterprise Application** in the console navigation tree.
2. On the first Preparing for application installation page, specify to install `JaxWSServicesSamples.ear`.
 - a. Click **Local file system** or **Remote file system** and specify the full path name of the `JaxWSServicesSamples.ear` file.
`app_server_root/samples/lib/JaxWSServicesSamples/JaxWSServicesSamples.ear`
 - b. Click **Next**.

3. On the second Preparing for application installation page, select the fast path option.
 - a. Select **Fast Path - Prompt only when additional information is required**.
 - b. Click **Next**.
4. Click **Next** on each panel until you reach the Summary panel.
Do not go directly from Step 1 to the Summary panel. You must click **Next** on each panel that has mandatory settings to enter values for those settings. Simply click **Next** to enter the default values. You optionally can change the values to suit your environment.
5. On the Summary panel, verify the cell, node, and server onto which the application modules will install, and then click **Finish**.
6. Examine the application installation progress messages.
If the application installs successfully, the message *Application JaxWSServicesSamples installed successfully* is displayed. Click **Save**. After the configuration changes are saved, you can see the name of the application in the list of deployed applications on the Enterprise applications page accessed by clicking **Applications** → **Application Types** → **WebSphere enterprise applications** in the console navigation tree.
If the application does not install successfully, read the messages to identify why the installation failed. Correct problems with the server or application and try installing the application again.

Results

The **JaxWSServicesSamples** application is in the list of deployed applications on the Enterprise applications page.

What to do next

After the application installs successfully, do the following:

1. Start the application.
On the Enterprise applications page, select the check boxes beside **JaxWSServicesSamples**, and then click **Start**.

2. Test the application. Point your Web browser at:

`http://localhost:9080/wssamplesei/demo`

If the localhost address does not load, substitute the host name (IP address) of the computer for localhost; for example, `http://9.22.33.44:9080/wssamplesei/demo`.

If you have another WebSphere Application Server installation on your machine, the server port number is likely not 9080. See the Ports table in the administrative console to find the `WC_defaulthost` server port number. Click **Servers** → **Server Types** → **WebSphere application servers** → *server1* → **Ports**. The Ports table lists the important ports:

Port name	Description
WC_adminhost	Port used to open an unsecure administrative console in the URL <code>http://host_name:administrative_port/ibm/console</code>
WC_adminhost_secure	Port used to open a secure administrative console in the URL <code>http://host_name:administrative_port/ibm/console</code>
WC_defaulthost	Port used to test running applications in the URL <code>http://host_name:server_port/servlet_name</code>
WC_defaulthost_secure	Port used to securely test running applications in the URL <code>http://host_name:server_port/servlet_name</code>

You can also view the running Web Services Sample in a Web browser open on the Samples Gallery at `http://localhost:9080/WSSamples`. As needed, substitute the host name (IP address) of the computer for localhost and the port name for 9080. After installation of the Web Services Sample, click **Installed**

Installing enterprise modules with JSR-88

You can install Java Platform, Enterprise Edition (Java EE) modules on an application server provided by a WebSphere Application Server product using the Java EE Application Deployment API specification (JSR-88).

Before you begin

JSR-88 defines standard application programming interfaces (APIs) to enable deployment of Java EE applications and stand-alone modules to Java EE product platforms. The Java EE Application Deployment specification Version 1.1 is available at <http://java.sun.com/j2ee/tools/deployment/reference/docs/index.html> as part of the Java 2 Platform, Enterprise Edition (J2EE) 1.4 Application Server Developer Release.

Read about JSR-88 and APIs used to manage applications at <http://java.sun.com/j2ee/tools/deployment/>.

About this task

JSR-88 defines a contract between a tool provider and a platform that enables tools from multiple vendors to configure, deploy and manage applications on any Java EE product platform. The tool provider typically supplies software tools and an integrated development environment (IDE) for developing and assembly of Java EE application modules. The Java EE platform provides application management functions that deploy, undeploy, start, stop, and otherwise manage Java EE applications.

WebSphere Application Server is a Java EE specification-compliant platform that implements the JSR-88 APIs. Complete the following steps to deploy (install) Java EE modules on an application server provided by the WebSphere Application Server platform.

1. Code a Java program that can access the JSR-88 DeploymentManager class for the product.
 - a. Write code that finds the JAR manifest attribute J2EE-DeploymentFactory-Implementation-Class. Under JSR-88, your code finds the DeploymentFactory using the JAR manifest attribute J2EE-DeploymentFactory-Implementation-Class. The following product application management JAR files contain this attribute and provide support:

Environment	JAR file containing the manifest attribute
Application server	<i>app_server_root</i> /plugins/com.ibm.ws.admin.services.jar
Application client	<i>app_client_root</i> /plugins/com.ibm.ws.j2ee.client.jar
Thin application client	<i>app_client_root</i> /runtimes/com.ibm.ws.admin.client_7.0.0.jar

After your code finds the DeploymentFactory, the deployment tool can create an instance of the WebSphere DeploymentFactory and register the instance with its DeploymentFactoryManager.

Example code for the application server environment follows. The example code requires that you use the development kit shipped with the product or use the pluggable client for deployment of stand-alone modules. See *WebSphere Application Server detailed system requirements* at <http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921> for information on supported development kits.

```
import javax.enterprise.deploy.shared.factories.DeploymentFactoryManager;
import javax.enterprise.deploy.spi.DeploymentManager;
import javax.enterprise.deploy.spi.factories.DeploymentFactory;
import java.util.jar.JarFile;
import java.util.jar.Manifest;

// Get the DeploymentFactory implementation class from the MANIFEST.MF file.
File jsr88Jar = new File(wasHome + "/plugins/com.ibm.ws.admin.services.jar");
JarFile jarFile = new JarFile(jsr88Jar);
Manifest manifest = jarFile.getManifest();
```

```

Attributes attributes = manifest.getMainAttributes();
String key = "J2EE-DeploymentFactory-Implementation-Class";
String className = attributes.getValue(key);
// Get an instance of the DeploymentFactoryManager
DeploymentFactoryManager dfm = DeploymentFactoryManager.getInstance();

// Create an instance of the WebSphere Application Server DeploymentFactory.
Class deploymentFactory = Class.forName(className);
DeploymentFactory deploymentFactoryInstance =
    (DeploymentFactory) deploymentFactory.newInstance();

// Register the DeploymentFactory instance with the DeploymentFactoryManager.
dfm.registerDeploymentFactory(deploymentFactoryInstance);

// Provide WebSphere Application Server URL, user ID, and password.
// For more information, see the step that follows.
wsDM = dfm.getDeploymentManager(
    "deployer:WebSphere:myserver:8880", null, null);

```

- b. Write code that accesses the DeploymentManager instance for the product The product URL for deployment has the format

```
"deployer:WebSphere:host:port"
```

The example in the previous step, "deployer:WebSphere:myserver:8880", tries to connect to host *myserver* at port *8880* using the SOAP connector, which is the default.

The URL for deployment can have an optional parameter *connectorType*. For example, to use the RMI connector to access *myserver*, code the URL as follows:

```
"deployer:WebSphere:myserver:2809?connectorType=RMI"
```

2. Optional: Code a Java program that can customize or deploy Java EE applications or modules using the JSR-88 support provided by the product.
3. Start the deployed Java EE applications or stand-alone Java EE modules using the JSR-88 API used to start applications or modules.

What to do next

Test the deployed applications or modules. For example, point a Web browser at the URL for a deployed application and examine the performance of the application. If necessary, update the application.

Customizing modules using DConfigBeans

You can configure Java Platform, Enterprise Edition (Java EE) applications or standalone modules during deployment using the DConfigBean class in the Java EE Application Deployment API specification (JSR-88).

Before you begin

This topic assumes that you are deploying (installing) Java EE modules on an application server provided by the product using the WebSphere Application Server support for JSR-88.

Read about the JSR-88 specification and using the DConfigBean class at <http://java.sun.com/j2ee/tools/deployment/>.

About this task

The DConfigBean class in JSR-88 provides JavaBeans-based support for platform-specific configuration of Java EE applications and modules during deployment. Your code can inspect DConfigBean instances to get platform-specific configuration attributes. The DConfigBean instances provided by WebSphere Application Server contain a single attribute which has an array of `java.util.Hashtable` objects. The hashtable entries contain configuration attributes, for which your code can get and set values.

1. Write code that installs Java EE modules on an application server using JSR-88.
2. Write code that accesses DConfigBeans generated by the product during JSR-88 deployment. You (or a deployer) can then customize the accessed DConfigBeans instances. The following pseudocode shows how a Java EE tool provider can get DConfigBean instance attributes generated by the product during JSR-88 deployment and set values for the attributes.

```
import javax.enterprise.deploy.model.*;
import javax.enterprise.deploy.spi.*;
{
DeploymentConfiguration dConfig = ___; // Get from DeploymentManager
DDBeanRoot ddRoot = ___; // Provided by J2EE tool

// Obtain root bean.
DConfigBeanRoot dcRoot = dConfig.getDConfigBeanRoot(dr);

// Configure DConfigBean.
configureDCBean (dcRoot);
}

// Get children from DConfigBeanRoot and configure each child.
method configureDCBean (DConfigBean dcBean)
{
// Get DConfigBean attributes for a given archive.
BeanInfo bInfo = Introspector.getBeanInfo(dcBean.getClass());
IndexedPropertyDescriptor ipDesc =
    (IndexedPropertyDescriptor)bInfo.getPropertyDescriptors()[0];

// Get the 0th table.
int index = 0;
Hashtable tbl = (Hashtable)
    ipDesc.getIndexedReadMethod().invoke
        (dcBean, new Object[]{new Integer(index)});

while (tbl != null)
{
// Iterate over the hashtable and set values for attributes.

// Set the table back into the DCBean.
ipDesc.getIndexedWriteMethod().invoke
    (dcBean, new Object[]{new Integer(index), tbl});

// Get the next entry in the indexed property
tbl = (Hashtable)
    ipDesc.getIndexedReadMethod().invoke
        (dcBean, new Object[]{new Integer(++index)});
}
}
```

Enterprise application collection

Use this page to view and manage enterprise applications.

This page lists installed enterprise applications. System applications, which are central to the product, are not shown in the list because users cannot edit them. Examples of system applications include *isclite*, *managementEJB* and *filetransfer*.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications**.

To view the values specified for an application's configuration, click the application name in the list. The displayed application settings page shows the values specified. On the settings page, you can change existing configuration values and link to additional console pages that assist you in configuring the application.

To manage an installed enterprise application, enable the **Select** check box beside the application name in the list and click a button:

Button	Resulting action
Start	Attempts to run the application. After the application starts up successfully, the state of the application changes to <i>Started</i> if the application starts up on all deployment targets, else the state changes to <i>Partial Start</i> .
Stop	Attempts to stop the processing of the application. After the application stops successfully, the state of the application changes to <i>Stopped</i> if the application stops on all deployment targets, else the state changes to <i>Partial Stop</i> .
Install	Opens a wizard that helps you deploy an application or a module such as a .jar, .war, .sar or .rar file onto a server or a cluster.
Uninstall	Deletes the application from the product configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed after the configuration is saved and synchronized with the nodes.
Update	Opens a wizard that helps you update application files deployed on a server. You can update the full application, a single module, a single file, or part of the application. If a new file or module has the same relative path as a file or module already existing on the server, the new file or module replaces the existing file or module. If the new file or module does not exist on the server, it is added to the deployed application.
Remove File	Deletes a file of the deployed application or module. Remove File deletes a file from the configuration repository and from the file system of all nodes where the file is installed.
Export	Accesses the Export Application EAR files page, which you use to export an enterprise application to an EAR file at a location of your choice. Use the Export action to back up a deployed application and to preserve its binding information.
Export DDL	Accesses the Export Application DDL files page, which you use to export DDL files (Table.ddl) in the EJB modules of an enterprise application to a location of your choice.
Export File	Accesses the Export a file from an application page, which you use to export a file of an enterprise application or module to a location of your choice. If the browser does not prompt for a location to store the file, click File → Save as and specify a location to save the file that is shown in the browser.

These buttons are not available when you access this page from an application server settings page. When this page is accessed from an application server settings page, it is entitled the Installed applications page.


When security is enabled, a separate application list is shown for each of your administrative roles. Supported roles include monitor, configurator, operator, administrator, deployer, and administrative security manager. For example, when you have the administrator role, the statement “You can administer the following resources” is shown followed by a list of applications that you can administer.






Name

Specifies the name of the installed (or deployed) application. Application names must be unique within a cell and cannot contain an unsupported character.

Application Status

Indicates whether the application deployed on the application server is started, stopped, or unknown.

	Started	Application is running.
---	----------------	-------------------------

	Partial Start	Application is in the process of changing from a <i>Stopped</i> state to a <i>Started</i> state. Application is starting to run but is not fully running yet. Or, it cannot fully start because a server mapped to one or more application modules is stopped.
	Stopped	Application is not running.
	Partial Stop	Application is in the process of changing from a <i>Started</i> state to a <i>Stopped</i> state. Application has not stopped running yet.
	Unknown	Status cannot be determined.
	Pending	Status is temporarily unknown pending an event that a user did not initiate, such as pending an asynchronous call.
	Not applicable	Application does not provide information as to whether it is running.

The status of an application on a Web server is always **Unknown**.

Startup order

Specifies the order in which applications are started when the server starts. The application with the lowest startup order is started first.

This table column is available only when this page is accessed from an application server settings page; thus when this page is entitled the Installed applications page.

Enterprise application settings

Use this page to configure an enterprise application.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name*.

If you have a JAX-WS Web service application installed, you also can click **Services** → **Service providers** → *service_name* or **Services** → **Service clients** → *service_name*.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Name

Specifies a logical name for the application. An application name must be unique within a cell and cannot contain an unsupported character.

An application name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

Data type String

Application reference validation

Specifies whether the product examines the application references specified during application installation or updating and, if validation is enabled, warns you of incorrect references or fails the operation.

An application typically refers to resources using data sources for container managed persistence (CMP) beans or using resource references or resource environment references defined in deployment descriptors. The validation checks whether the resource referred to by the application is defined in the scope of the deployment target of that application.

The resource can be defined on the server, its node, cell or the cluster if the server belongs to a cluster. Select **Don't validate** for no resource validation, **Issue warnings** for warning messages about incorrect resource references, or **Stop installation if validation fails** to stop operations that fail as a result of incorrect resource references.

This **Application reference validation** setting is the same as the **Validate input off/warn/fail** field on the application installation and update wizards.

Data type String
Default Issue warnings

Configuring enterprise application files

You can change the configuration of a Java Platform, Enterprise Edition (Java EE) application or module deployed on a server.

Before you begin

You can change the contents of and deployment descriptors for an application or module before deployment, such as in an assembly tool. However, it is assumed that the module is already deployed on a server.

About this task

Changing an application or module configuration consists of one or more of the following:

- Changing the settings of the application or module.
- Removing a file from an application or module.
- Updating the application or its modules.

This topic describes how to change the settings of an application or module using the administrative console.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

- View current settings of the application or module.

Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* to access the enterprise application settings page.

Many application or module settings are available on other console pages that you can access by clicking links on the settings page for the enterprise application. For detailed information on the settings and allowed values, examine the online help for the console pages. When you installed the application or module, you specified most of the settings values.

- Map each module of your application to a target server.
Specify the application servers or Web servers onto which to install modules of your application.
- Change how quickly your application starts compared to other applications or to the server.
- Configure the use of binary files.
- Change how your application or Web modules use class loaders.
- Map a virtual host for each Web module of your application. Configuring virtual hosts provides information on virtual hosts.
- Change application bindings or other settings of the application or module.
 1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → **application_name** → **property_or_item_name**. From the application settings page, you can access console pages for further configuring of the application or module.
 2. Change the values for settings as needed, and click **OK**.
- Optional: Configure the application so it does not start automatically when the server starts. By default, an installed application starts when the server on which the application resides starts. You can configure the target mapping for the application so the application does not start automatically when the server starts. To start the application, you must then start it manually.
- If the installed application or module uses a resource adapter archive (RAR file), ensure that the **Classpath** setting for the RAR file enables the RAR file to find the classes and resources that it needs. Examine the **Classpath** setting on the console Resource adapter settings page.

Results

The application or module configuration is changed. The application or standalone Web module is restarted so the changes take effect.

What to do next

Save changes to your administrative configuration.

Application bindings

Before an application that is installed on an application server can start, all enterprise bean (EJB) references and resource references defined in the application must be bound to the actual artifacts (enterprise beans or resources) defined in the application server.

When defining bindings, you specify Java Naming and Directory Interface (JNDI) names for the referenceable and referenced artifacts in an application. The `jndiName` values specified for artifacts must be qualified lookup names. An example referenceable artifact is an EJB defined in an application. An example referenced artifact is an EJB or a resource reference used by the application.

Binding definitions are stored in the `ibm-xxx-bnd.xml` or `ibm-xxx-bnd.xmi` files of an application. Version 7.0 binding definitions support files with the suffix of XML for EJB 3.0 and Web 2.5 modules. Modules earlier than Java EE 5 continue to use binding definition files with the suffix of XMI as in previous versions of WebSphere Application Server. The `xxx` can be `ejb-jar`, `web`, `application` or `application-client`.

Note: For EJB 3.0 modules, you do not need to specify JNDI binding names for each of the home or business interfaces on your enterprise beans. If you do not explicitly assign bindings, the EJB container assigns default bindings. Further, binding definitions are stored in `ibm-ejb-jar-bnd.xml`. See EJB 3.0 application bindings overview.

This topic provides the following information about bindings:

- “Times when bindings can be defined”
- “Required bindings”
- “Other bindings that might be needed” on page 276

Times when bindings can be defined

You can define bindings at the following times:

- During application development

An application developer can create binding definitions in `ibm-xxx-bnd.xml` files for EJB 3.0 and Web 2.5 modules and in `ibm-xxx-bnd.xmi` files for pre-Java EE 5 modules. The application developer can create the files using a tool such as an IBM Rational developer tool or, for EJB 3.0 or Web 2.5 modules, using an XML editor or text editor. The developer then gives an enterprise application (.ear file) complete with bindings to an application assembler or deployer. When assembling the application, the assembler does not modify the bindings. Similarly, when installing the application onto a server supported by WebSphere Application Server, the deployer does not modify or override the bindings or generate default bindings unless changes to the bindings are necessary for successful deployment of the application.

- During application assembly

An application assembler can define bindings in annotations or deployment descriptors of an application. Java EE 5 modules contain annotations in the source code. To declare an annotation, an application assembler precedes a keyword with an @ character (“at” sign). Bindings for pre-Java EE 5 modules are specified in the **WebSphere Bindings** section of a deployment descriptor editor. Modifying the deployment descriptors might change the binding definitions in the `ibm-xxx-bnd.xmi` files created when developing an application. After defining the bindings, the assembler gives the application to a deployer. When installing the application onto a server supported by WebSphere Application Server, the deployer does not modify or override the bindings or generate default bindings unless changes to the bindings are necessary to deploy the application.

- During application installation

An application deployer or server administrator can modify the bindings when installing the application onto a server supported by WebSphere Application Server using the administrative console. New binding definitions can be specified on the installation wizard pages.

Also, a deployer or administrator can select to generate default bindings during application installation. Selecting **Generate default bindings** during application installation instructs the product to fill in incomplete bindings in the application with default values. Existing bindings are not changed.

Note: You cannot define or override bindings during application installation for application clients. You must define bindings for application client modules during assembly and store the bindings in the `ibm-application-client-bnd.xmi` file.

- During configuration of an installed application

After an application is installed onto a server supported by WebSphere Application Server, an application deployer or server administrator can modify the bindings by changing values in administrative console pages such as those accessed from the settings page for the enterprise application.

Required bindings

Before an application can be successfully deployed, bindings must be defined for references to the following artifacts:

EJB JNDI names

For each EJB 2.1 or earlier enterprise bean (EJB), you must specify a JNDI name. The name is used to bind an entry in the global JNDI name space for the EJB home object. An example JNDI

name for a *Product* EJB in a *Store* application might be `store/ejb/Product`. The binding definition is stored in the `META-INF/ibm-ejb-jar-bnd.xml` file.

If a deployer chooses to generate default bindings when installing the application, the installation wizard assigns EJB JNDI names having the form `prefix/EJB_name` to incomplete bindings. The default prefix is `ejb`, but can be overridden. The `EJB_name` is as specified in the deployment descriptor `<ejb-name>` tag.

During and after application installation, EJB JNDI names can be specified on the Provide JNDI names for beans panel. After installation, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **EJB JNDI names** in the administrative console.

You do not need to specify JNDI binding names for each of the EJB 3.0 home or business interfaces on your enterprise beans because the EJB container assigns default bindings. See EJB 3.0 application bindings overview.

Data sources for entity beans

Entity beans such as container-managed persistence (CMP) beans store persistent data in data stores. With CMP beans, an EJB container manages the persistent state of the beans. You specify which data store a bean uses by binding an EJB module or an individual enterprise bean to a data source. Binding an EJB module to a data source causes all entity beans in that module to use the same data source for persistence.

An example JNDI name for a *Store* data source in a *Store* application might be `store/jdbc/store`. For modules earlier than Java EE 5, the binding definition is stored in IBM binding files such as `ibm-ejb-jar-bnd.xml`. A deployer can also specify whether authentication is handled at the container or application level.

WebSphere Application Server Version 7.0 supports CMP beans in EJB 2.x or 1.x modules. Version 7.0 does not support CMP beans in EJB 3.0 modules.

If a deployer chooses to generate default bindings when installing the application, the install wizard generates the following for incomplete bindings:

- For EJB 2.x .jar files, connection factory bindings based on the JNDI name and authorization information specified
- For EJB 1.1 .jar files, data source bindings based on the JNDI name, data source user name and password specified

The generated bindings provide default connection factory settings for each EJB 2.x .jar file and default data source settings for each EJB 1.1 .jar file in the application being installed. No bean-level connection factory bindings or data source bindings are generated unless they are specified in the custom strategy rule supplied during default binding generation.

During and after application installation, you can map data sources to 2.x entity beans on the 2.x CMP bean data sources panel and on the 2.x entity bean data sources panel. After installation, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* in the administrative console, then select **2.x CMP bean data sources** or **2.x entity bean data sources**. You can map data sources to 1.x entity beans on the Map data sources for all 1.x CMP beans panel and on the Provide default data source mapping for modules containing 1.x entity beans panel. After installation, access console pages similar to those for 2.x CMP beans, except click links for 1.x CMP beans.

Backend ID for EJB modules

If an EJB .jar file that defines CMP beans contains mappings for multiple backend databases, specify the appropriate backend ID that determines which persister classes are loaded at run time.

Specify the backend ID during application installation. You cannot select a backend ID after the application is installed onto a server.

To enable backend IDs for individual EJB modules:

1. During application installation, select **Deploy enterprise beans** on the Select installation options panel. Selecting **Deploy enterprise beans** enables you to access the Provide options to perform the EJB Deploy panel.
2. On the Provide options to perform the EJB Deploy panel, set the database type to "" (null).

During application installation, if you select **Deploy enterprise beans** on the Select installation options panel and specify a database type for the EJB deployment tool on the Provide options to perform the EJB Deploy panel, previously defined backend IDs for all of the EJB modules are overwritten by the chosen database type.

The default database type is DB2UDB_V81.

The EJB deployment tool does not run during installation of EJB 3.0 modules.

EJB references

An enterprise bean (EJB) reference is a logical name used to locate the home interface of an enterprise bean. EJB references are specified during deployment. At run time, EJB references are bound to the physical location (global JNDI name) of the enterprise beans in the target operational environment. EJB references are made available in the java:comp/env/ejb Java naming subcontext.

The product assigns default JNDI values for or automatically resolves incomplete EJB 3.0 reference targets.

For each EJB 2.1 or earlier EJB reference, you must specify a JNDI name. An example JNDI name for a *Supplier* EJB reference in a *Store* application might be store/ejb/Supplier. The binding definition is stored in IBM binding files such as ibm-ejb-jar-bnd.xmi. When the referenced EJB is also deployed in the same application server, you can specify a server-scoped JNDI name. But if the referenced EJB is deployed on a different application server or if ejb-ref is defined in an application client module, then you should specify the global cell-scoped JNDI name.

If a deployer chooses to generate default bindings when installing the application, the install wizard binds EJB references as follows: If an <ejb-link> is found, it is honored. If the ejb-name of an EJB defined in the application matches the ejb-ref name, then that EJB is chosen. Otherwise, if a unique EJB is found with a matching home (or local home) interface as the referenced bean, the reference is resolved automatically.

During and after application installation, you can specify EJB reference JNDI names on the Map EJB references to beans panel. After installation, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **EJB references** in the administrative console.

Note: To enable EJB reference targets to resolve automatically if the references are from EJB 2.1 or earlier modules or from Web 2.3 or earlier modules, select **Generate default bindings** on the Preparing for application installation panel or select **Allow EJB reference targets to resolve automatically** on the Select installation options, Map EJB references to beans, or EJB references console panels.

For more information, refer to EJB references .

Resource references

A resource reference is a logical name used to locate an external resource for an application. Resource references are specified during deployment. At run time, the references are bound to the physical location (global JNDI name) of the resource in the target operational environment. Resource references are made available as follows:

Resource reference type	Subcontext declared in
Java DataBase Connectivity (JDBC) data source	java:comp/env/jdbc
JMS connection factory	java:comp/env/jms
JavaMail connection factory	java:comp/env/mail

Uniform Resource Locator (URL) connection factory	java:comp/env/url
---	-------------------

For each resource reference, you must specify a JNDI name. If a deployer chooses to generate default bindings when installing the application, the install wizard generates resource reference bindings derived from the `<res-ref-name>` tag, assuming that the `java:comp/env` name is the same as the resource global JNDI name.

During application installation, you can specify resource reference JNDI names on the Map resource references to references panel. Specify JNDI names for the resources that represent the logical names defined in resource references. You can optionally specify login configuration name and authentication properties for the resource. After specifying authentication properties, click **OK** to save the values and return to the mapping step. Each resource reference defined in an application must be bound to a resource defined in your WebSphere Application Server configuration. After installation, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Resource references** in the administrative console to access the Resource references panel.

Virtual host bindings for Web modules

You must bind each Web module to a specific virtual host. The binding informs a Web server plug-in that all requests that match the virtual host must be handled by the Web application. An example virtual host to be bound to a *Store* Web application might be `store_host`. The binding definition is stored in IBM binding files such as `WEB-INF/ibm-web-bnd.xml`.

If a deployer chooses to generate default bindings when installing the application, the install wizard sets the virtual host to `default_host` for each `.war` file.

During and after application installation, you can map a virtual host to a Web module defined in your application. On the Map virtual hosts for Web modules panel, specify a virtual host. The port number specified in the virtual host definition is used in the URL that is used to access artifacts such as servlets and JavaServer Pages (JSP) files in the Web module. For example, an external URL for a Web artifact such as a JSP file is `http://host_name:virtual_host_port/context_root/jsp_path`. After installation, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Virtual hosts** in the administrative console.

Message-driven beans

For each message-driven bean, you must specify a queue or topic to which the bean will listen. A message-driven bean is invoked by a Java Messaging Service (JMS) listener when a message arrives on the input queue that the listener is monitoring. A deployer specifies a listener port or JNDI name of an activation specification as defined in a connector module (`.rar` file) under **WebSphere Bindings** on the Beans page of an assembly tool EJB deployment descriptor editor. An example JNDI name for a listener port to be used by a *Store* application might be `StoreMdbListener`. The binding definition is stored in IBM bindings files such as `ibm-ejb-jar-bnd.xml`.

If a deployer chooses to generate default bindings when installing the application, the install wizard assigns JNDI names to incomplete bindings.

- For EJB 2.0 or later message-driven beans deployed as JCA 1.5-compliant resources, the install wizard assigns JNDI names corresponding to activationSpec instances in the form `eis/MDB_ejb-name`.
- For EJB 2.0 or later message-driven beans deployed against listener ports, the listener ports are derived from the message-driven bean `<ejb-name>` tag with the string `Port` appended.

During application installation using the administrative console, you can specify a listener port name or an activation specification JNDI name for every message-driven bean on the Bind listeners for message-driven beans panel. A listener port name must be provided when using the JMS providers: Version 5 default messaging, WebSphere MQ, or generic. An activation specification must be provided when the application's resources are configured using the default messaging provider or any generic J2C resource adapter that supports inbound messaging. If neither is specified, then a validation error is displayed after you click **Finish** on the Summary

panel. Also, if the module containing the message-driven bean is deployed on a 5.x deployment target and a listener port is not specified, then a validation error is displayed after you click **Next**.

After application installation, you can specify JNDI names and configure message-driven beans on console pages under **Resources** → **JMS** → **JMS providers** or under **Resources** → **Resource adapters**. For more information, refer to Choosing a messaging provider.

Note: You can only bind message driven-beans that are defined in an EJB 3.0 module to an activation specification. See EJB 3.0 application bindings overview.

Message destination references

A message destination reference is a logical name used to locate an enterprise bean in an EJB module that acts as a message destination. Message destination references exist only in J2EE 1.4 and later artifacts such as--

- J2EE 1.4 application clients
- EJB 2.1 projects
- 2.4 Web applications

If multiple message destination references are associated with a single message destination link, then a single JNDI name for an enterprise bean that maps to the message destination link, and in turn to all of the linked message destination references, is collected during deployment. At run time, the message destination references are bound to the administered message destinations in the target operational environment.

If a message destination reference and a message-driven bean are linked by the same message destination, both the reference and the bean should have the same destination JNDI name. When both have the same name, only the destination JNDI name for the message-driven bean is collected and applied to the corresponding message destination reference.

If a deployer chooses to generate default bindings when installing the application, the install wizard assigns JNDI names to incomplete message destination references as follows: If a message destination reference has a <message-destination-link>, then the JNDI name is set to ejs/message-destination-linkName. Otherwise, the JNDI name is set to eis/message-destination-refName.

Other bindings that might be needed

Depending on the references in and artifacts used by your application, you might need to define bindings for references and artifacts not listed in this topic.

Configuring application startup

You can configure the startup behavior of an application. The values set affect how quickly an application starts and what occurs when an application starts.

Before you begin

This topic assumes that your application or module is already deployed on a server.

This topic also assumes that your application or module is configured to start automatically when the server starts. By default, an installed application starts when the server on which the application resides starts.

About this task

This topic describes how to change the settings of an application or module using the administrative console.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Startup behavior** in the console navigation tree.

The Startup behavior settings page is displayed.

2. Specify the startup order for the application.

If your application starts automatically when its server starts, the value for **Startup order** on the Startup behavior settings page specifies the order in which applications are started when the server starts. The application with the lowest startup order, or starting weight, is started first. For example, specify 1 for **Startup order** for applications that you want started first, specify 2 for applications to be started next, and so.

Note: For Session Initiation Protocol (SIP) applications, the `<load-on-startup>` tag in the sip.xml file affects the order in which applications are started. The value that you set for **Startup order** on the Startup behavior settings page determines the importance or weight of an application within a composition of SIP applications. For example, for the most important SIP application within a SIP application composition, specify 1 for **Startup order**. For the next most important SIP application within the composition, specify 2 for **Startup order**, and so on.

3. Specify whether the application must initialize fully before its server is considered started.

If your application starts automatically when its server starts, **Launch application before server completes startup** specifies whether the application must initialize fully before its server is considered started. Background applications can be initialized on an independent thread, thus allowing the server startup to complete without waiting for the application. This setting applies only if the application is run on a Version 6.0 or later application server.

4. Specify whether to create MBeans for resources such as servlets or JavaServer Pages (JSP) files within an application when the application starts.

The default for **Create MBeans for resources** is to create MBeans.

Results

The application or module configuration is changed. The application or standalone Web module is restarted so the changes take effect.

What to do next

Save changes to your administrative configuration.

Startup behavior settings

Use this page to configure when an application starts compared to other applications and to the server, and to configure whether MBeans for resources are created when an application starts.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Startup behavior**.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Startup order:

Specifies the order in which applications are started when the server starts. The startup order is like a starting weight. The application with the lowest starting weight is started first.

Note: For Session Initiation Protocol (SIP) applications, the `<load-on-startup>` tag in the `sip.xml` file affects the order in which servlets within applications are started. The value that you set for **Startup order** on this Startup behavior console page determines the importance or weight of an application within a composition of SIP applications. For example, for the most important SIP application within a SIP application composition, specify 1 for **Startup order**. For the next most important SIP application within the composition, specify 2 for **Startup order**, and so on. For more information, see the JSR 116 specification.

Data type	Integer
Default	1
Range	0 to 2147483647

Launch application before server completes startup:

Specifies whether the application must initialize fully before the server starts.

The default setting of `false` indicates that server startup will not complete until the application starts.

A setting of `true` informs the product that the application might start on a background thread and thus server startup might continue without waiting for the application to start. Thus, the application might not be ready for use when the application server starts.

This setting applies only if the application is run on a Version 6.0 or later application server.

Data type	Boolean
Default	<code>false</code>

Create MBeans for resources:

Specifies whether to create MBeans for various resources, such as servlets or JavaServer Pages (JSP) files, within an application when the application starts. The default is to create MBeans.

Data type	Boolean
Default	<code>true</code>

Configuring binary location and use

You can designate where binary files (binaries) used by your application reside, whether the product distributes binaries for you automatically, and otherwise configure the use of binaries.

Before you begin

This topic assumes that your application or module is already deployed on a server.

About this task

This topic describes how to change the settings of an application or module using the administrative console.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node

where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Application binaries** in the console navigation tree. The Application binaries page is displayed.
2. Specify the directory to hold the application binaries.
The default is `${APP_INSTALL_ROOT}/cell_name`, where the `${APP_INSTALL_ROOT}` variable is `profile_root/installedApps`. For example:
`profile_root/installedApps/cell_name`
Refer to “Application binary settings” for a detailed description of the **Location (full path)** setting.
3. Specify the bindings, extensions, and deployment descriptors that an application server uses.
By default, an application server uses the bindings, extensions, and deployment descriptors located with the application deployment document, the `deployment.xml` file.
To specify that the application server use the bindings, extensions, and deployment descriptors located in the application archive (EAR) file, select **Use configuration information in binary**. Select this setting for applications installed on 6.x or later deployment targets. This setting is not valid for applications installed on 5.x deployment targets.
4. Specify whether the product distributes application binaries automatically to other nodes on the cell.
By default, **Enable binary distribution, expansion and cleanup post uninstallation** is selected and binaries are distributed automatically.
If you disable this option, then you must ensure that the application binaries are expanded appropriately in the destination directories of all nodes where the application runs.

Note: If you disable this option and you do not copy and expand the application binaries to the nodes, a later saving of the configuration or manual synchronization does not move the application binaries to the nodes for you.
5. Specify access permissions for binaries.
 - a. Ensure that the **Enable binary distribution, expansion and cleanup post uninstallation** option is enabled. That option must be enabled to specify access permissions for binaries.
 - b. For **File permissions**, specify a string that defines access permissions for binaries that are expanded in the named location.
You can specify file permissions in the text field. You can also set some of the commonly used file permissions by selecting them from the multiple-selection list. List selections overwrite file permissions set in the text field.
For details on **File permissions**, refer to “Application binary settings.”
6. Click **OK**.

Results

The application or module configuration is changed. The application or stand-alone Web module is restarted so the changes take effect.

What to do next

Save changes to your administrative configuration.

Application binary settings

Use this page to configure the location and distribution of application binary files.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Application binaries**.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Location (full path):

Specifies the directory to which the enterprise application archive (EAR) file is installed. This **Location** setting is the same as the **Directory to install application** field on the application installation and update wizards.

By default, an EAR file is installed in the *profile_root/installedApps/cell_name/application_name.ear* directory.

Setting options include the following:

- Do not specify a value and leave the field empty.

The default value is *\${APP_INSTALL_ROOT}/cell_name*, where the *\${APP_INSTALL_ROOT}* variable is *profile_root/installedApps*. A directory having the EAR file name of the application being installed is appended to *\${APP_INSTALL_ROOT}/cell_name*. Thus, if you do not specify a directory, the EAR file is installed in the *profile_root/installedApps/cell_name/application_name.ear* directory.

- Specify a directory.

If you specify a directory, the application is installed in *specified_path/application_name.ear* directory. A directory having the EAR file name of the application being installed is appended to the path that you specified for **Directory to install application** when installing the application. For example, if you installed Clock.ear and specify *C:/myapps* on Windows machines, the application is installed in the *myapps/Clock.ear* directory. The *\${APP_INSTALL_ROOT}* variable is set to the specified path.

- Specify *\${APP_INSTALL_ROOT}/\${CELL}* for the initial installation of the application.

If you intend to export the application from one cell and later install the exported application on a different cell, specify the *{CELL}* variable for the initial installation of the application. For example, specify *\${APP_INSTALL_ROOT}/\${CELL}* for this setting. Exporting the application creates an enhanced EAR file that has the application and its deployment configuration. The deployment configuration retains the cell name of the initial installation in the destination directory unless you specify the *{CELL}* variable. Specifying the *{CELL}* variable ensures that the destination directory has the current cell name, and not the original cell name.

Note: If an installation directory is not specified when an application is installed on a single-server configuration, the application is installed in *\${APP_INSTALL_ROOT}/cell_name*. When the server is made a part of a multiple-server configuration (using the addNode utility), the cell name of the new configuration becomes the cell name of the deployment manager node. If the *-includeapps* option is used for the addNode utility, then the applications that are installed prior to the addNode operation still use the installation directory *\${APP_INSTALL_ROOT}/cell_name*. However, an application that is installed after the server is added to the network configuration uses the default installation directory *\${APP_INSTALL_ROOT}/network_cell_name*. To move the application to the *\${APP_INSTALL_ROOT}/network_cell_name* location upon running the addNode operation, explicitly specify the installation directory as *\${APP_INSTALL_ROOT}/\${CELL}* during installation. In such a case, the application files can always be found under *\${APP_INSTALL_ROOT}/current_cell_name*.

- If the application has been exported and you want to install the exported EAR file in a different cell or location, specify *\${APP_INSTALL_ROOT}/cell_name/application_name.ear* if you did not specify *\${APP_INSTALL_ROOT}/\${CELL}* for the initial installation.

The exported EAR file is an enhanced EAR file that has the application and its deployment configuration. The deployment configuration retains the value for **Directory to install application** that

was used for the previous installation of the application. Unless you specify a different value, the enhanced EAR file will be installed to the same directory as for the previous installation.

If you did not specify the `${CELL}` variable during the initial installation, the deployment configuration uses the cell name of the initial installation in the destination directory. If you are installing on a different cell, specify `${APP_INSTALL_ROOT}/cell_name/application_name.ear`, where `cell_name` is the name of the cell to which you want to install the enhanced EAR file. If you do not designate the current cell name, `cell_name` will be the original cell name even though you are installing the enhanced EAR file on a cell that has a different name.

- Specify an absolute path or a use pathmap variable.

You can specify an absolute path or use a pathmap variable such as `${MY_APPS}`. You can use a pathmap variable in any installation.

Data type	String
Units	Full path name

Use configuration information in binary:

Specifies whether the application server uses the binding, extensions, and deployment descriptors located with the application deployment document, the `deployment.xml` file (default), or those located in the EAR file.

The default (false) is to use the binding, extensions, and deployment descriptors located in `deployment.xml`. To use the binding, extensions, and deployment descriptors located in the EAR file, enable this setting (true).

This **Use configuration information in binary** setting is the same as the **Use binary configuration** field on the application installation and update wizards. Select this setting for applications installed on 6.x or later deployment targets only. This setting is not valid for applications installed on 5.x deployment targets.

Data type	Boolean
Default	false

Enable binary distribution, expansion and cleanup post uninstallation:

Specifies whether the product expands application binaries in the installation location during installation and deletes application binaries during uninstallation. The default is to enable application distribution. Application binaries for installed applications are expanded to the directory specified.

On single-server installations, the binaries are deleted when you uninstall and save changes to the configuration.

If you disable this option, then you must ensure that the application binaries are expanded appropriately in the destination directories of all nodes where the application runs.

Note: If you disable this option and you do not copy and expand the application binaries to the nodes, a later saving of the configuration or manual synchronization does not move the application binaries to the nodes for you.

This **Enable binary distribution, expansion and cleanup post uninstallation** setting is the same as the **Distribute application** field on the application installation and update wizards.

Data type	Boolean
Default	true

File permissions:

Specifies access permissions for application binaries for installed applications that are expanded to the directory specified.

The **Enable binary distribution, expansion and cleanup post uninstallation** option must be enabled to specify file permissions.

You can specify file permissions in the text field. You can also set some of the commonly used file permissions by selecting them from the multiple-selection list. List selections overwrite file permissions set in the text field.

You can set one or more of the following file permission strings in the list. Selecting multiple options combines the file permission strings.

Multiple-selection list option	File permission string set
Allow all files to be read but not written to	.*=755
Allow executables to execute	.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
Allow HTML and image files to be read by everyone	.*\.htm=755#.*\.html=755#.*\.gif=755#.*\.jpg=755

Instead of using the multiple-selection list to specify file permissions, you can specify a file permission string in the text field. File permissions use a string that has the following format:

file_name_pattern=permission#file_name_pattern=permission

where *file_name_pattern* is a regular expression file name filter (for example, *.**.jsp* for all JSP files), *permission* provides the file access control lists (ACLs), and *#* is the separator between multiple entries of *file_name_pattern* and *permission*. If *#* is a character in a *file_name_pattern* string, use *\#* instead.

If multiple file name patterns and file permissions in the string match a uniform resource identifier (URI) within the application, then the product uses the most stringent applicable file permission for the file. For example, if the file permission string is *.**.jsp=775#a.**.jsp=754*, then the *abc.jsp* file has file permission 754.

Note: Using regular expressions for file matching pattern compares an entire string URI against the specified file permission pattern. You must provide more precise matching patterns using regular expressions as defined by Java programming API. For example, suppose the following directory and file URIs are processed during a file permission operation:

1	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war
2	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp
3	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/META-INF/MANIFEST.MF
4	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/WEB-INF/classes/MyClass.class
5	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/mydir/MyClass2.class
6	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/META-INF

The file pattern matching results are:

- MyWarModule.war does not match any of the URIs
- *.*MyWarModule.war.** matches all URIs

- `.*MyWarModule.war$` matches only URI 1
- `.*\.\.jsp=755` matches only URI 2
- `.*META-INF.*` matches URIs 3 and 6
- `.*MyWarModule.war/.*/.*\..class` matches URIs 4 and 5

If you specify a directory name pattern for **File permissions**, then the directory permission is set based on the value specified. Otherwise, the **File permissions** value set on the directory is the same as its parent. For example, suppose you have the following file and directory structure:

```
/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp
```

and you specify the following file pattern string:

```
.*MyApp.ear$=755#.*\..jsp=644
```

The file pattern matching results are:

- Directory `MyApp.ear` is set to 755
- Directory `MyWarModule.war` is set to 755
- Directory `MyWarModule.war` is set to 755

Note: Regardless of the operation system, always use a forward slash (/) as a file path separator in file patterns.

Access permissions specified here are at the application level. You can also specify access permissions for application binaries in the node level configuration. The node level file permissions specify the maximum (most lenient) permissions that can be given to application binaries. Access permissions specified here at application level can only be the same as or more restrictive than those specified at the node level.

This setting is the same as the **File permission** field on the application installation and update wizards.

Data type String

Application build level:

Specifies an uneditable string that identifies the build version of the application.

Data type String

Configuring the use of class loaders by an application

You can configure whether your application and Web modules use their own class loaders to load classes or use different class loaders, as well as configure the reloading of classes when application files are updated. Class loaders enable an application to access repositories of available classes and resources.

Before you begin

This topic assumes that your application or module is already deployed on a server.

About this task

Class loaders affect whether your application and its modules find the resources that they need to run effectively. You can select whether your application and Web modules use their own class loaders to load classes, or use a parent class loader.

An application class loader groups enterprise bean (EJB) modules, shared libraries, resource adapter archives (RAR files), and dependency Java archive (JAR) files associated to an application. Dependency JAR files are JAR files that contain code which can be used by both enterprise beans and servlets.

An application class loader is the parent of a Web application archive (WAR) class loader. By default, a Web module has its own WAR class loader to load the contents of the Web module. The WAR class-loader policy value of an application class loader determines whether the WAR class loader or the application class loader is used to load the contents of the Web module.

You can also select whether classes are reloaded when application files are updated. For enterprise bean (EJB) modules or any non-Web modules, enabling class reloading causes the application server run time to stop and start the application to reload application classes. For Web modules such as servlets and JavaServer Pages (JSP) files, a Web container reloads a Web module only when the IBM extension reloadingEnabled in the ibm-web-ext.xmi file is set to true.

To configure use of class loaders by your application and Web modules, use the Class loading and update detection page of the administrative console.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Class loading and update detection** to access the settings page for an application class loader.
2. Specify whether to reload application classes when the application or its files are updated.
By default, class reloading is not enabled. Select **Override class reloading settings for Web and EJB modules** to choose to reload application classes. You might specify different values for EJB modules and for Web modules such as servlets and JavaServer Pages (JSP) files.
3. Specify the number of seconds to scan the application's file system for updated files.
The value specified for **Polling interval for updated files** takes effect only if class reloading is enabled. The default is the value of the reloading interval attribute in the IBM extension (META-INF/ibm-application-ext.xmi) file of the enterprise application (EAR file). You might specify different values for EJB modules and for Web modules such as servlets and JSP files.
To enable reloading, specify an integer value that is greater than zero (for example, 1 to 2147483647).
To disable reloading, specify zero (0).
4. Specify the class loader order for the application.
The application class loader order specifies whether the class loader searches in the parent class loader or in the application class loader first to load a class. The default is to search in the parent class loader before searching in the application class loader to load a class.
Select either of the following values for **Class loader order**:

Option	Description
Classes loaded with parent class loader first	Causes the class loader to search in the parent class loader first to load a class. This value is the standard for Development Kit class loaders and WebSphere Application Server class loaders.

Option	Description
Classes loaded with local class loader first (parent last)	Causes the class loader to search in the application class loader first to load a class. By specifying <code>Classes loaded with local class loader first (parent last)</code> , your application can override classes contained in the parent class loader. Note: Specifying the <code>Classes loaded with local class loader first (parent last)</code> value might result in <code>LinkageErrors</code> or <code>ClassCastException</code> messages if you have mixed use of overridden classes and non-overridden classes.

- Specify whether to use a single or multiple class loaders to load Web application archives (WAR files) of your application.

By default, Web modules have their own WAR class loader to load the contents of the `WEB-INF/classes` and `WEB-INF/lib` directories. The default WAR class loader value is `Class loader for each WAR file in application`, which uses a separate class loader to load each WAR file. Setting the value to `Single class loader for application` causes the application class loader to load the Web module contents as well as the EJB modules, shared libraries, RAR files, and dependency JAR files associated to the application. The application class loader is the parent of the WAR class loader.

Select either of the following values for **WAR class loader policy**:

Option	Description
Class loader for each WAR file in application	Uses a different class loader for each WAR file.
Single class loader for application	Uses a single class loader to load all of the WAR files in your application.

- Click **OK**.

Results

The application or module configuration is changed. The application or standalone Web module is restarted so the changes take effect.

What to do next

Save changes to your administrative configuration.

Class loading and update detection settings

Use this page to configure use of class loaders by an application.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Class loading and update detection**.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Override class reloading settings for Web and EJB modules:

Specifies whether to enable class reloading when application files are updated.

Select **Override class reloading settings for Web and EJB modules** to set reloadEnabled to true in the deployment.xml file for the application. If an application's class definition changes, the application server run time stops and starts the application to reload application classes.

Reloading settings in the deployment.xml file override the reloading settings for all Web and EJB modules that can be defined in ibm-web-ext.xmi and META-INF/ibm-application-ext.xmi files.

For JavaServer Pages (JSP) files in a Web module, a Web container reloads JSP files only when the IBM extension jspReloadingEnabled in the jspAttributes of the ibm-web-ext.xmi file is set to true. You can enable JSP reloading during deployment on the JSP Reload Options panel.

Data type	Boolean
Default	false

Polling interval for updated files:

Specifies the number of seconds to scan the application's file system for updated files. The default is the value of the reloading interval attribute in the IBM extension (META-INF/ibm-application-ext.xmi) file of the EAR file.

This **Polling interval for updated files** setting is the same as the **Reload interval in seconds** field on the application installation and update wizards.

To enable reloading, specify a value greater than zero (for example, 1 to 2147483647). To disable reloading, specify zero (0). The range is from 0 to 2147483647.

The reloading interval attribute takes effect only if class reloading is enabled.

Data type	Integer
Units	Seconds
Default	3

Class loader order:

Specifies whether the class loader searches in the parent class loader or in the application class loader first to load a class. The standard for development kit class loaders and WebSphere Application Server class loaders is `Classes loaded with parent class loader first`. By specifying `Classes loaded with local class loader first (parent last)`, your application can override classes contained in the parent class loader, but this action can potentially result in `ClassCastException` or `LinkageErrors` if you have mixed use of overridden classes and non-overridden classes.

The options are `Classes loaded with parent class loader first` and `Classes loaded with local class loader first (parent last)`. The default is to search in the parent class loader before searching in the application class loader to load a class.

For your application to use the default configuration of Jakarta Commons Logging in WebSphere Application Server, set this application class loader mode to `Classes loaded with parent class loader first`. For your application to override the default configuration of Jakarta Commons Logging in WebSphere Application Server, your application must provide the configuration in a form supported by Jakarta Commons Logging and this class loader mode must be set to `Classes loaded with local class loader first (parent last)`. Also, to override the default configuration, set the class loader mode for each Web module in your application so that the correct logger factory loads.

Data type	String
Default	Classes loaded with parent class loader first

WAR class loader policy:

Specifies whether to use a single class loader to load all WAR files of the application or to use a different class loader for each WAR file.

The options are Class loader for each WAR file in application and Single class loader for application. The default is to use a separate class loader to load each WAR file.

Data type	String
Default	Class loader for each WAR file in application

Manage modules settings

Use this panel to specify deployment targets where you want to install the modules that are contained in your application. Modules can be installed on the same deployment target or dispersed among several deployment targets.

On single-server products, a deployment target can be an application server or Web server.

To view this administrative console panel, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Manage modules**. This panel is similar to the Map modules to servers panel on the application installation and update wizards.

On this panel, each **Module** must map to one or more targets, identified under **Server**. To change a mapping:

1. In the list of mappings, select each module that you want mapped to the same target or targets.
2. From the **Clusters and servers** list, select one or more targets. Select only appropriate deployment targets for a module. You cannot install modules that use WebSphere Application Server Version 7.x features on a Version 6.x or 5.x target server.

Use the Ctrl key to select multiple targets. For example, to have a Web server serve your application, press the Ctrl key and then select an application server and the Web server together. The product generates the plug-in configuration file, plugin-cfg.xml, for that Web server based on the applications which are routed through it.

3. Click **Apply**.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

If you accessed this Manage modules panel from a console enterprise application page for an already installed application, you can also use this panel to view and manage modules in your application.

To view the values specified for a module configuration, click the module name in the list. The displayed module settings page shows the values specified. On the settings page, you can change existing configuration values and link to additional console pages that assist you in configuring the module.

To manage a module, select the module name in the list and click a button:

Button	Resulting action
Remove	Removes the selected module from the deployed application. The module is deleted from the application in the configuration repository and also from all of the nodes where the application is installed and running or expected to run.
Update	Opens a wizard that helps you update modules in an application. If a module has the same URI as a module already existing in the application, the new module replaces the existing module. If the new module does not exist in the application, it is added to the deployed application.
Remove File	Deletes a file from a module of a deployed application.
Export File	Accesses the Export a file from an application page, which you use to export a file of an enterprise application or module to a location of your choice. If the browser does not prompt for a location to store the file, click File → Save as and specify a location to save the file that is shown in the browser.

Clusters and servers

Lists the names of available deployment targets. This list is the same for every application that is installed in the cell.

From this list, select only appropriate deployment targets for a module. You must install an application, enterprise bean (EJB) module, Session Initiation Protocol (SIP) module (SAR), or Web module on a Version 7.x target under any of the following conditions:

- The module supports Java Platform, Enterprise Edition (Java EE) 5.
- The module calls a 7.x runtime application programming interface (API).
- The module uses a 7.x product feature.

If a module supports J2EE 1.4, then you must install the module on a Version 6.x or 7.x deployment target. Modules that call a 6.1.x API or use a 6.1.x feature can be installed on a 6.1.x or 7.x deployment target. Modules that call a 6.0.x API or use a 6.0.x feature can be installed on a 6.0.x, 6.1.x or 7.x deployment target. Modules that require 6.1.x feature pack functionality can be installed on a 6.1.x deployment target that has been enabled with that feature pack or on a 7.x deployment target.

You can install an application or module developed for a Version 5.x product on a 5.x, 6.x or 7.x deployment target.

Module

Specifies the name of a module in the installed (or deployed) application.

URI

Specifies the location of the module relative to the root of the application (EAR file).

Module type

Specifies the type of module, for example, a Web module or EJB module.

This setting is shown on the Manage modules panel accessed from a console enterprise application page.

Server

Specifies the name of each deployment target to which the module currently is mapped.

To change the deployment targets for a module, select one or more targets from the **Clusters and servers** list and click **Apply**. The new mapping replaces the previous mapping.

Mapping modules to servers

Each module of a deployed application must be mapped to one or more target servers. The target server can be an application server or Web server.

Before you begin

You can map modules of an application or stand-alone Web module to one or more target servers during or after application installation using the console. This topic assumes that the module is already installed on a server and that you want to change the mappings.

Before you change a mapping, check the deployment targets. You must specify an appropriate deployment target for a module. Modules that use Version 7.x features cannot be installed onto Version 6.x or 5.x target servers.

About this task

During application installation, different deployment targets might have been specified.

You use the Manage modules panel of the administrative console to view and change mappings. This panel is displayed during application installation using the console and, after the application is installed, can be accessed from the enterprise application settings page.

On the Manage modules panel, specify target servers where you want to install the modules contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that will serve as routers for requests to your application. The plug-in configuration file, `plugin-cfg.xml`, for each Web server is generated based on the applications which are routed through it.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Manage modules** in the console navigation tree.
The Manage modules panel is displayed.
2. Examine the list of mappings.
Ensure that each **Module** entry is mapped to one or more targets, identified under **Server**.
3. Change a mapping as needed.
 - a. Select each module that you want mapped to the same targets.
In the list of mappings, select check boxes for the modules.
 - b. From the **Clusters and servers** list, select one or more targets.
Select only appropriate deployment targets for a module. You cannot install modules that use WebSphere Application Server Version 7.x features on a Version 6.x or 5.x target server.
Use the Ctrl key to select multiple targets. For example, to have a Web server serve your application, use the Ctrl key to select an application server and the Web server together to have the `plugin-cfg.xml` plug-in configuration file for that Web server generated based on the applications that are routed through it.
 - c. Click **Apply**.
4. Repeat steps 2 and 3 until each module maps to the desired targets.
5. Click **OK**.

Results

The application or module configurations are changed. The application or stand-alone Web module is restarted so the changes take effect.

Example

To install an application that has modules which support Java Platform, Enterprise Edition (Java EE) 5 to two servers, do the following:

1. Click the **Select All** icon to select all of the modules in the application.
2. While pressing Ctrl, select two Version 7 application servers from the **Clusters and servers** list.
3. Click **Apply**.
4. Click **OK**.

What to do next

Save changes to your administrative configuration.

Mapping virtual hosts for Web modules

A virtual host must be mapped to each Web module of a deployed application. Web modules can be installed on the same virtual host or dispersed among several virtual hosts.

Before you begin

You can map a virtual host to a Web module during or after application installation using the console. This article assumes that the Web module is already installed on a server and that you want to change the mappings.

Before you change a mapping, check the virtual hosts definitions. You can install a Web module on any defined virtual host. To view information on previously defined virtual hosts, click **Environment** → **Virtual hosts** in the administrative console. Virtual hosts enable you to associate a unique port with a module or application. The aliases of a virtual host identify the port numbers defined for that virtual host. A port number specified in a virtual host alias is used in the URL that is used to access artifacts such as servlets and JavaServer Pages (JSP) files in a Web module. For example, the alias `myhost:8080` is the `host_name:port_number` portion of the URL `http://myhost:8080/servlet/snoop`.

About this task

During application installation, a virtual host other than the one you want mapped to your Web module might have been specified.

The default virtual host setting usually is `default_host`, which provides several port numbers through its aliases:

- 80** An internal, insecure port used when no port number is specified
- 9080** An internal port
- 9443** An external, secure port

Unless you want to isolate your Web module from other modules or resources on the same node (physical machine), `default_host` is a suitable virtual host for your Web module.

In addition to `default_host`, the product provides `admin_host`, which is the virtual host for the administrative console system application. `admin_host` is on port 9060. Its secure port is 9043. Do not select `admin_host` unless the Web module relates to system administration.

Use the Virtual hosts page of the administrative console to view and change mappings. This page is displayed during application installation using the console and, after the application is installed, can be accessed from the settings page for an enterprise application.

On the Virtual hosts page, specify a virtual host for each Web module. Web modules of an application can be installed on the same virtual host or on different virtual hosts.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Virtual hosts** in the console navigation tree. The Virtual hosts page is displayed.
2. Examine the list of mappings. Ensure that each **Web module** entry has the desired virtual host mapped to it, identified under **Virtual host**.
3. Change the mappings as needed.
 - a. Select each Web module that you want mapped to a particular virtual host. In the list of mappings, place a check mark in the **Select** check boxes beside the Web modules.
 - b. From the **Virtual host** drop-down list, select the desired virtual host. If you selected more than one virtual host in step 1:
 - 1) Expand **Apply Multiple Mappings**.
 - 2) Select the desired virtual host from the **Virtual host** drop-down list.
 - 3) Click **Apply**.
4. Repeat steps 2 and 3 until a desired virtual host is mapped to each Web module.
5. Click **OK**.

Results

The application or Web module configurations are changed. The application or standalone Web module is restarted so the changes take effect.

What to do next

After mapping virtual hosts, do the following:

1. Regenerate the plug-in configuration file.
 - a. Click **Servers** → **Server Types** → **Web servers**.
 - b. Select the Web server for which you want to generate a plug-in.
 - c. Click **Generate Plug-in**.
2. Save changes to your administrative configuration.

Virtual hosts settings

Use this panel to specify virtual hosts for Web modules contained in your application. Web modules can be installed on the same virtual host or dispersed among several virtual hosts.

To view this administrative console panel, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Virtual hosts**. This panel is the same as the **Map virtual hosts for Web modules** panel on the application installation and update wizards.

On this panel, each Web module must map to a previously defined virtual host, identified under **Virtual host**. You can see information on previously defined virtual hosts by clicking **Environment** → **Virtual hosts** in the administrative console. Virtual hosts enable you to associate a unique port with a module or application. The aliases of a virtual host identify the port numbers defined for that virtual host. A port number specified in a virtual host alias is used in the URL that is used to access artifacts such as servlets and JavaServer Pages (JSP) files in a Web module. For example, the alias `myhost:8080` is the `host_name:port_number` portion of the URL `http://myhost:8080/servlet/snoop`.

The default virtual host setting usually is `default_host`, which provides several port numbers through its aliases:

- 80** An internal, insecure port used when no port number is specified

- 9080 An internal port
- 9443 An external, secure port

Unless you want to isolate your Web module from other modules or resources on the same node (physical machine), `default_host` is a suitable virtual host for your Web module.

In addition to `default_host`, the product provides `admin_host`, which is the virtual host for the administrative console system application. `admin_host` is on port 9060. Its secure port is 9043. Do not select `admin_host` unless the Web module relates to system administration.

To change a mapping:

1. In the list of mappings, select the **Select** check box beside each Web module that you want mapped to a particular virtual host.
2. From the **Virtual host** drop-down list, select the desired virtual host. If you selected more than one virtual host in step 1:
 - a. Expand **Apply Multiple Mappings**.
 - b. Select the desired virtual host from the **Virtual Host** drop-down list.
 - c. Click **Apply**.
3. Click **OK**.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

Web module:

Specifies the name of a Web module in the application that you are installing or that you are viewing after installation.

Virtual host:

Specifies the name of the virtual host to which the Web module is currently mapped.

Expanding the drop-down list displays a list of previously defined virtual hosts. To change a mapping, select a different virtual host from the list.

Do not specify the same virtual host for different Web modules that have the same context root and are deployed on targets belonging to the same node even if the Web modules are contained in different applications. Specifying the same virtual host causes a validation error.

Mapping properties for a custom login or trusted connection configuration

Use this page to view and manage the mapping properties for a custom login configuration or a trusted connection configuration.

To access the administrative console panel, complete the following steps:

1. Click **Applications > Application types > WebSphere enterprise applications > *application_name***.
2. From Enterprise JavaBeans Properties, click **Map data sources for all 2.x CMP beans**.
3. For container authorization, modify the authorization type by selecting your Enterprise JavaBeans(EJB) module and selecting **Container** from the Resource authorization menu.
4. Click **Apply**.

5. From Specify authentication method, select **Use custom login configuration** or **Use trusted connections** and the name of the application login configuration.
6. Select the name of your EJB module.
7. Click **Apply**.
8. Click **Mapping properties** in the Resource authorization column. This property is not available until after you click **Apply** in the previous step.

Name

Specifies the name for the mapping property.

Do not use the MAPPING_ALIAS property name because the name is reserved by the product.

Value

Specifies the value paired with the specified name.

Description

Specifies additional information about the name and value pair.

Viewing deployment descriptors

A deployment descriptor is an extensible markup language (XML) file that specifies configuration and container options for an application or module.

Before you begin

This topic assumes that you have installed an application or module on a server and that you want to view its deployment descriptor.

About this task

When you create a Java 2 Platform, Enterprise Edition (J2EE) application or module in an assembly tool, the assembly tool creates deployment descriptor files for the application or module. Java Platform, Enterprise Edition (Java EE) 5 applications or modules might use annotations instead of deployment descriptors.

After an application or module is installed on a server, you can view its deployment descriptor in the administrative console. You cannot view Java EE 5 annotations.

Unless an application supports Java EE 5, an enterprise archive (EAR) file must contain an application.xml file. The application.xml identifies each module of an application. A Java EE 5 application is not required to provide an application.xml file in the EAR file. When an application.xml file does not exist, the product examines the Java archive (JAR) file contents to determine whether the JAR file is an enterprise bean (EJB) module or an application client module. A JAR file should not contain more than one deployment descriptor in it. When an ejb-jar.xml file is found in a JAR file, the product considers it an EJB module. If an ejb-jar.xml file is not found and an application-client.xml is found, the product considers the JAR file to be an application client module. If both ejb-jar.xml and application-client.xml files exist in the JAR file, the product might consider a JAR file intended to be an application client module to be an EJB module or a JAR file intended to be an EJB module to be an application client module. A JAR file should not contain more than one kind of deployment descriptor.

1. Access a deployment descriptor view.

Click the navigational option stated in **Accessing a console view** to view the deployment descriptor for a given module:

Module	Deployment descriptor file	Accessing a console view
Enterprise application	application.xml	Applications → Application Types → WebSphere enterprise applications → <i>application_name</i> → View deployment descriptor
Web application	WEB-INF/web.xml	Applications → Application Types → WebSphere enterprise applications → <i>application_name</i> → Manage modules → <i>module_name</i> → View deployment descriptor
	WEB-INF/portlet.xml	Applications → Application Types → WebSphere enterprise applications → <i>application_name</i> → Manage modules → <i>module_name</i> → View portlet deployment descriptor
Enterprise bean	ejb-jar.xml	Applications → Application Types → WebSphere enterprise applications → <i>application_name</i> → Manage modules → <i>module_name</i> → View deployment descriptor
Application client	application-client.xml	No console view
Web service	webservices.xml	Applications → Application Types → WebSphere enterprise applications → <i>application_name</i> → Manage modules → <i>module_name</i> > <ul style="list-style-type: none"> • View Web services client deployment descriptor extension • View Web services server deployment descriptor • View Web services server deployment descriptor extension <p>Viewing Web services deployment descriptors in the administrative console describes the views.</p>
Resource adapter	ra.xml	Resources → Resource Adapters → Resource adapters → <i>module_name</i> → View deployment descriptor

2. Click **Expand All** to view the deployment descriptor contents.

Results

The deployment descriptor for the application or module is displayed.

Example

The deployment descriptor for the product DefaultApplication follows:

```
<application id="Application_ID" >
  <display-name> DefaultApplication.ear</display-name>
  <description> This is the IBM WebSphere Application Server Default Application.</description>
  <module id="WebModule_1" >
    <web>
      <web-uri> DefaultWebApplication.war</web-uri>
      <context-root> /</context-root>
    </web>
  </module>
  <module id="EjbModule_1" >
    <ejb> Increment.jar</ejb>
  </module>
  <security-role id="SecurityRole_1204342979281" >
    <description> All Authenticated users role.</description>
    <role-name> All Role</role-name>
  </security-role>
</application>
```

What to do next

After displaying a deployment descriptor on the console page, do the following:

1. Examine the deployment descriptor contents, including any configurations that it has for bindings, security roles, references to other resources, or Java Naming and Directory Interface (JNDI) names. For example, examine the JAR files of your Java EE 5 module to ensure that each JAR file does not contain more than one kind of deployment descriptor. If a JAR file contains more than one kind of deployment descriptor, proceed to the next step and remove the extraneous deployment descriptor. Thus, if both `ejb-jar.xml` and `application-client.xml` files exist in a JAR file, remove the deployment descriptor that your module does not need.
2. Change a deployment descriptor as needed.
You can edit a deployment descriptor file manually. However, it is preferable to edit a deployment descriptor using the console or in an assembly tool deployment descriptor editor to ensure that the deployment descriptor has valid properties and that its references contain appropriate values.

If your EJB 3.0 or Web 2.5 module does not have a `metadata-complete` attribute or the `metadata-complete` attribute is set to `false`, you can instruct the product to write the entire module deployment descriptor, including deployment information from annotations, to XML format. On the Metadata for modules page, select **metadata-complete attribute**.

Metadata for module settings

Use this page to instruct a Java Platform, Enterprise Edition (Java EE) 5 enterprise bean (EJB) or Web module deployment descriptor to ignore annotations that specify deployment information.

To view this administrative console panel, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Metadata for modules**. This panel is the same as the Metadata for modules panel on the application installation and update wizards.

Note: If your application contains EJB 3.0 or Web 2.5 modules, you can select to lock the deployment descriptor of one or more of the EJB 3.0 or Web 2.5 modules on the Metadata for modules panel. If you set the `metadata-complete` attribute to `true` and lock deployment descriptors, the product writes the complete module deployment descriptor, including deployment information from annotations, to XML format.

Annotations are a standard mechanism of adding metadata to Java classes. You can use metadata to simplify development and deployment of Java EE 5 artifacts. Prior to the introduction of Java language annotations, deployment descriptors were the standard mechanism used by Java EE components. These deployment descriptors were mapped to XML format, which facilitated their persistence. When you select to lock deployment descriptors, the product merges Java EE 5 annotation-based metadata with XML-based metadata for those annotations that are mapped to existing deployment descriptor attributes.

Module

Specifies the name of a module in the installed (or deployed) application.

Data type String

URI

Specifies the location of the module relative to the root of the application (EAR file).

Data type String

metadata-complete attribute

Specifies whether to write the complete module deployment descriptor, including deployment information from annotations, to extensible markup language (XML) format.

The default is not to write out a module deployment descriptor.

If your EJB 3.0 or Web 2.5 module does not have a metadata-complete attribute or the metadata-complete attribute is set to false, you can select a check box and instruct the product to write out a module deployment descriptor.

After you select a check box, you cannot deselect (clear) the check box and the module is no longer shown in the list of modules on this page. If you select all of the check boxes, the link to this page is no longer shown on the enterprise application settings page.

Data type	Boolean
Default	false (deselected)

Starting or stopping enterprise applications

You can start an application that is not running (has a status of *Stopped*) or stop an application that is running (has a status of *Started*).

Before you begin

This topic assumes that the Java Platform, Enterprise Edition (Java EE) application is installed on a server. By default, the application starts automatically when the server starts.

About this task

You can start and stop applications manually using the following:

- Administrative console
- startApplication and stopApplication attributes of the AdminControl object with the wsadmin tool
- startApplication and stopApplication administrative jobs of the AdminTask.submitJob -jobType object with the wsadmin tool
- Java programs that use ApplicationManager or AppManagement MBeans

This topic describes how to use the administrative console to start or stop an application.

Note: This topic applies to applications that do not contain Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) or Service Component Architecture (SCA) service providers. To stop or start applications that contain JAX-WS or SCA service providers, use the Service providers page accessed by clicking **Services** → **Service providers**. To start a service provider application, select a service and click **Start Application**. To stop a service provider application, select a service and click **Stop Application**. Then, on the Stop application page, click **OK** to stop all modules in the application, including other services such as enterprise beans and servlets.

1. Go to the Enterprise applications page. Click **Applications** → **Application Types** → **WebSphere enterprise applications** in the console navigation tree.
2. Select the check box for the application you want started or stopped.
3. Click a button:

Option	Description
Start	Runs the application and changes the state of the application to <i>Started</i> . The status is changed to <i>partially started</i> if not all servers on which the application is deployed are running.
Stop	Stops the processing of the application and changes the state of the application to <i>Stopped</i> .

To restart a running application, select the application you want to restart, click **Stop** and then click **Start**.

Results

The status of the application changes and a message stating that the application started or stopped displays at the top the page.

What to do next

You can configure an application so it does not start automatically when the server on which it resides starts. You then start the application manually using options described in this article.

If you want your application to start automatically when its server starts, you can adjust values that control how quickly the application or its server starts:

1. Go the settings page for your enterprise application. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Startup behavior**.
2. Specify a different value for **Startup order**.
This setting specifies the order in which applications are started when the server starts. The default value is 1 in a range from 0 to 2147483647. The application with the lowest starting weight is started first.
3. Specify a different value for **Launch application before server completes startup**.
This setting specifies whether the application must initialize fully before its server starts. The default value of `false` prevents the server from starting completely until the application starts. To reduce the amount of time it takes to start the server, you can set the value to `true` and have the application start on a background thread, thus allowing server startup to continue without waiting for the application.
4. Save the changes to the application configuration.

Disabling automatic starting of applications

You can enable and disable the automatic starting of an application. By default, an installed application starts automatically when the server on which the application resides starts.

Before you begin

This topic assumes that the application is installed on an application server and that the application starts automatically when the server starts.

This topic also assumes that you mapped the installed application to a server and that you have an administrative role with an authority higher than monitor.

About this task

You might want an application to run only after you start it manually and not to run every time after the server starts. The target mapping for an application controls whether an application starts automatically when the server starts or requires you to start the application manually.

You must have an administrative role with an authority higher than monitor to change the automatic starting setting.

1. Go to the Target specific application status page for your application.
Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Target specific application status**.
2. Select the target server on which the application resides.
3. Click **Disable Auto Start**.
4. Save changes to the administrative configuration.

Results

The application does not start when its server starts. You must start the application manually.

What to do next

To enable automatic starting of the application, do the following:

1. On the Target specific application status page for the application, select the target on which the application resides.
2. Click **Enable Auto Start**.
3. Save changes to the configuration.

Target specific application status

Use this page to view mappings of deployed applications or modules to servers.

Also use this page to enable or disable the automatic starting of an application when the server on which the application resides starts.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Target specific application status**.

When security is enabled, a separate application list is shown for each of your administrative roles. Supported roles include monitor, configurator, operator, administrator, deployer, and administrative security manager. For example, when you have the administrator role, the statement “You can administer the following resources” is shown followed by a list of servers that you can administer.

Target

States the name of the target server to which the application or module maps. You specify the target on the Manage modules page accessed from the settings for an application.

Node

Specifies the node name if the target is a server.

Version

Specifies the version level of the target. The target can be a Version 7.x, 6.x or 5.x deployment target.

A *7.x deployment target* is a server with all members on a WebSphere Application Server Version 7.0 or later product.

A *6.x deployment target* is a server with all members on a WebSphere Application Server Version 6 product.

A *5.x deployment target* is a server with at least one member on a WebSphere Application Server Version 5 product.

An application, enterprise bean (EJB) module, Session Initiation Protocol (SIP) module (SAR), or Web module must be installed on a Version 7.x target under any of the following conditions:

- The module supports Java Platform, Enterprise Edition (Java EE) 5.
- The module calls a 7.x runtime application programming interface (API).
- The module uses a 7.x product feature.

If a module supports Java 2 Platform, Enterprise Edition (J2EE) 1.4, then you must install the module on a Version 6.x or 7.x deployment target. Modules that call a 6.1.x API or use a 6.1.x feature can be installed on a 6.1.x or 7.x deployment target. Modules that call a 6.0.x API or use a 6.0.x feature can be installed

on a 6.0.x, 6.1.x or 7.x deployment target. Modules that require 6.1.x feature pack functionality can be installed on a 7.x deployment target or on a 6.1.x deployment target that has been enabled with that feature pack.

If JavaServer Pages (JSP) precompilation, EJB deployment (ejbdeploy), or Web Services deployment (wsdeploy) are enabled, then you can deploy applications to only those targets that have same product version as the deployment manager. If applications are targeted to servers that have an earlier version than the deployment manager, then you cannot deploy to those targets. Thus, if JSP precompilation, ejbdeploy, or wsdeploy are enabled, then you must deploy the application on a 6.1.x or 7.x target.

You can install an application or module developed for a Version 5.x product on a 7.x, 6.x or 5.x deployment target.

Similarly, a resource adapter (connector) module, or RAR file, developed for a Version 5.x product can reside on a 7.x, 6.x or 5.x target, provided the module does not support Java Cryptography Architecture (JCA) 1.5 and does not call any 7.x or 6.x runtime application programming interfaces (APIs). If the module supports JCA 1.5 or calls a 7.x or 6.x API, then the module must reside on a 7.x or 6.x target.

Auto Start







Specifies whether the application modules installed on the target server are started (or enabled) when the server starts. This setting specifies the initial state of application modules. A **Yes** value indicates that the corresponding modules are enabled and thus are accessible when the server starts. A **No** value indicates that the corresponding modules are not enabled and thus are not accessible when the server starts.

By default, Auto Start is enabled. Thus, by default an installed application starts automatically when the server on which the application resides starts.

If you have an administrative role with an authority higher than monitor, you can enable and disable the automatic starting of the application. To disable the automatic starting of the application, enable the **Select** check box beside the target server and click **Disable Auto Start**. When automatic starting is disabled, the application does not start when its server starts. To enable the automatic starting of the application, select the target and click **Enable Auto Start**.

Application Status

Indicates whether the application deployed on the application server is started, stopped, or unknown.

	Started	Application is running.
	Partial Start	Application is in the process of changing from a <i>Stopped</i> state to a <i>Started</i> state. Application is starting to run but is not fully running yet. The application might be in the Partial Start state because one of its application servers is not started.
	Stopped	Application is not running.
	Partial Stop	Application is in the process of changing from a <i>Started</i> state to a <i>Stopped</i> state. Application has not stopped running yet.
	Unknown	Status cannot be determined. An application with an unknown status might, in fact, be running but have an unknown status because the server running the administrative console cannot communicate with the server running the application.
	Pending	Status is temporarily unknown pending an event that a user did not initiate, such as pending an asynchronous call.
	Not applicable	Application does not provide information as to whether it is running.

The status of an application on a Web server is always **Unknown**.

Exporting enterprise applications

You can export an enterprise application to a location of your choice.

About this task

Exporting applications enables you to back up your applications and preserve binding information for the applications. You might export your applications before updating installed applications or migrating to a later version of the product.

To export applications, use the **Export** button on the Enterprise applications page. Using **Export** produces an enhanced enterprise archive (EAR) file that contains the application as well as the deployment configuration. The *deployment configuration* consists of the deployment.xml and other configuration files that control the application behavior on a deployment target.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** in the console navigation tree to access the Enterprise applications page.
2. Select the check box beside the application and click **Export**.
3. On the Export application EAR files page, click on the link to download the exported EAR file.
4. Use the browser dialogue to specify a location at which to save the exported EAR file.
User profile QEJBSVR must have *WX authority to the directory and at least *X authority to all directories in the path specified for the location.
5. Click **Back** to return to the Enterprise applications page.

Results

The file containing binding information is exported to the specified node and directory, and has the name *enterprise_application_name.ear*.

Using the **Export** button to export applications does not export any manual changes that were made to applications in the installedApps directory. To export those changes, you must copy and move the application files manually.

What to do next

You can edit your exported enhanced EAR file and then reinstall it. By default, installation expands an EAR file in the *profile_root/installedApps/cell_name* directory. If you specified the \$(CELL) variable for **Directory to install application** on the Select installation options panel of the application installation wizard when you first installed the application, the *cell_name* directory is the current cell name.

To reinstall the enhanced EAR file, do either of the following:

- Use the **Update** operation available from the Enterprise applications page to upgrade the existing application installation.
The **Update** operation adds the application files to the *profile_root/installedApps/cell_name* directory, where *cell_name* is the current cell name or the name of the cell that you specified for **Directory to install application** when you first installed the application on a deployment target. The **Directory to install application** setting is on the Select installation options panel of the application installation wizard. If you specified the \$(CELL) variable for **Directory to install application** when you first installed the application, the *cell_name* directory is the current cell name.
- Use the **Applications** → **New application** → **New Enterprise Application** operation to install the exported EAR file.

If you specified the \$(CELL) variable for **Directory to install application** when you first installed the application, the *cell_name* directory is the current cell name. That is, if the file is originally installed on

Cell1 with \$(CELL) variable in the destination directory and you reinstall the enhanced EAR file on Cell2, the *cell_name* directory is Cell2, the current cell name.

If the \$(CELL) variable was not specified for the first installation, using **New Enterprise Application** to reinstall an enhanced EAR file installs the application in the *cell_name* directory of the exported application. That is, if the application is originally installed on and exported from Cell1 and you reinstall the enhanced EAR file on Cell2, the *cell_name* directory is Cell1. The enhanced EAR file expands in the Cell1 directory even though the current cell name is Cell2. By default, the application destination directory contains Cell1 in its path because the deployment.xml file in the exported application has Cell1 in it.

If you exported the application from Cell1 and did not specify the \$(CELL) variable when first installing the application, and you want to install the enhanced EAR file on a different cell, deselect **Process embedded configuration** on the Select installation options panel of the application installation wizard to expand the enhanced EAR file in the current cell name directory, which is not Cell1.

Exporting enterprise application files

You can export individual files of a Java Platform, Enterprise Edition (Java EE) application or module.

Before you begin

This topic assumes that you have installed an application or module on a server and that you want to export a file in the application or module.

About this task

Exporting a file in a deployed application or module downloads the file to a location of your choice.

To export a file using the administrative console, use **Export File**.

To export an entire application, use **Export**. For information on **Export**, see “Exporting enterprise applications” on page 300. The exported enterprise archive (EAR) file contains application configuration data as well as the application.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** in the administrative console navigation tree to access the Enterprise applications page.
2. Place a check mark in the check box beside the application and click **Export File**. A drop-down list of exportable files is displayed.
3. Select a file from the list and click **Export**. A dialog in which you select a target location is displayed. If the browser does not prompt for a location to store the file, click **File** → **Save as** and specify a location to save the file that is shown in the browser.
4. Specify the location to which to download the file.
User profile QEJBSVR must have *WX authority to the directory and at least *X authority to all directories in the path specified for the location.

Results

The file is downloaded to the specified location.

What to do next

Click **Back** to return to the Enterprise applications page.

Exporting DDL files

You can export data definition language (DDL) files in the enterprise bean (EJB) modules of an application.

About this task

Exporting DDL (Table.ddl) files in the EJB modules of an application downloads the DDL files to a location of your choice.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** in the administrative console navigation tree to access the Enterprise applications page.
2. Place a check mark in the check box beside the application and click **Export DDL**. If the application has no DDL files in any of its EJB modules, then the message *No DDL files were found* is displayed at the top of the page. If the application has DDL files in its EJB modules, then a page listing DDL files in the format *application_name.ear/_module.jar_Table.ddl* is displayed.
3. Click on a file in the list and specify the location to which to download the file.
User profile QEJBSVR must have *WX authority to the directory and at least *X authority to all directories in the path specified for the location.

Note: For Firefox browsers, right-click the file name, select **Save Link As**, and specify the location to which to download the file.

Mozilla browsers might display the contents of the Table.ddl file instead of saving the file to disk. To save the file, edit the **Helper Application** preference settings of the Mozilla browser by adding a new type for DDL and specifying that you want to save DDL files to disk. That is, set MIME type = ddl and Extension = ddl.

Results

The product downloads the DDL file to the specified location.

Updating enterprise application files

You can update Java Platform, Enterprise Edition (Java EE) application files deployed on a server.

Before you begin

Update your Java EE application or modules and reassemble them using an assembly tool. Typical tasks include adding or editing assembly properties, adding or importing modules into an application, and adding enterprise beans, Web components, and files.

Also, determine whether the updated files can be installed to your deployment targets. WebSphere Application Server Version 7.x and later supports Java EE 5 enterprise applications and modules.

If you are deploying Java EE 5 modules, ensure that the deployment target supports Version 7.x. You can deploy Java EE 5 modules to Version 7.x servers only. You cannot deploy Java EE 5 modules to Version 6.x deployment targets.

About this task

Updating consists of adding a new file or module to an installed application, or replacing or removing an installed application, file or module. After replacement of a full application, the old application is uninstalled. After replacement of a module, file or partial application, the old installed module, file or partial application is removed from the installed application.

1. Determine which method to use to update your application files. The product provides several ways to update modules.
2. Update the application files using
 - Administrative console
 - wsadmin scripts
 - Java application programming interfaces
 - WebSphere rapid deployment of Java EE applications

In some situations, you can update applications or modules without restarting the application server using hot deployment. Do not use hot deployment unless you are an experienced user and are updating applications in a development or test environment.
3. Start the deployed application files using
 - Administrative console
 - wsadmin startApplication
 - Java programs that use ApplicationManager or AppManagement MBeans

What to do next

Save the changes to your administrative configuration.

Next, test the application. For example, point a Web browser at the URL for a deployed application (typically `http://hostname:9060/Web_module_name`, where *hostname* is your valid Web server and 9060 is the default port number) and examine the performance of the application. If the application does not perform as desired, edit the application configuration, then save and test it again.

Ways to update enterprise application files

You can update Java Platform, Enterprise Edition (Java EE) application files deployed on a server in several ways.

Table 5. Ways to update application files

Option	Method	Comments	Starting after update
Administrative console update wizard See “Updating enterprise applications with the console” on page 305. To remove a single file from a Java EE application or module, see “Removing enterprise files” on page 318.	Briefly, do the following: 1. Go to the Enterprise Applications page. Click Applications → Application Types → WebSphere enterprise applications in the console navigation tree. 2. Select the application to update and click Update . 3. On the Preparing for application update page, identify the application, module or files to update and click Next . 4. Complete steps in the update wizard and click Finish .	On the Preparing for application update page: • Use Full application to update an .ear file. • Use Single module to update a .war, .sar, enterprise bean .jar, or connector .rar file. • Use Single file to update a file other than an .ear, .war, .sar, EJB .jar, or .rar file. • Use Partial application to update or remove multiple files.	On the Enterprise applications page, select the updated application and click Start .
wsadmin scripts	Use the update command or the updateInteractive command in a script or at a command prompt.	“Getting started with scripting” in the Using the administrative clients PDF provides an overview of wsadmin.	Start the application using the invoke command and the startApplication attribute.

Table 5. Ways to update application files (continued)

Option	Method	Comments	Starting after update
Java application programming interfaces	Update deployed applications by completing the steps in "Managing applications through programming" in the Using the administrative clients PDF.	Update an application in the following ways: <ul style="list-style-type: none"> • Update the entire application • Add to, replace or delete multiple files in an application • Add a module to an application • Update a module in an application • Delete a module in an application • Add a file to an application • Update a file in an application • Delete a file in an application 	<ul style="list-style-type: none"> • Invoke the AdminApp <i>startApplication</i> command. • Invoke the <i>startApplication</i> method on an ApplicationManager MBean using AdminControl.
Rapid deployment tools See topics under Rapid deployment of J2EE applications.	Briefly, do the following: <ol style="list-style-type: none"> 1. Update your J2EE application files. 2. Set up the rapid deployment environment. 3. Create a free-form project. 4. Launch a rapid deployment session. 5. Drop your updated application files into the free-form project. 	Rapid deployment tools offer the following advantages: <ul style="list-style-type: none"> • You do not need to assemble your J2EE application files prior to deployment. • You do not need to use other installation tools mentioned in this table to deploy the files. 	Use any of the above options to start the application. Clicking Start on the Enterprise applications page is the easiest option.
Hot deployment and dynamic reloading	Briefly, do the following: <ol style="list-style-type: none"> 1. Update your application (.ear), Web module (.war), enterprise bean .jar or HTTP plug-in configuration file. 2. Follow instructions in Hot deployment and dynamic reloading to update your file. 	If you are new to WebSphere Application Server, use the administrative console to update applications. That option is easier. Hot deployment and dynamic reloading is more difficult to complete. You must directly manipulate the application or module file on the server where the application is deployed.	Use any of the above options to start the application. Clicking Start on the Enterprise applications page is the easiest option.

You can update .ear, enterprise bean .jar, Web module .war, Session Initiation Protocol (SIP) module (.sar), connector .rar, application client .jar, and any other files used by an installed application.

If the application is updated while it is running, WebSphere Application Server automatically stops the application, updates the application logic and restarts the application. If the application does not start automatically, start it manually using one of the **Starting** options. For more information on the restarting of updated applications, refer to "Fine-grained recycle behavior" in *IBM WebSphere Developer Technical Journal: System management for WebSphere Application Server V6 -- Part 5 Flexible options for updating deployed applications.*

Updating enterprise applications with the console

Updating enterprise applications consists of adding a new file or module to an installed Java Platform, Enterprise Edition (Java EE) application, or replacing or removing an installed application, file or module.

Before you begin

Before you update the application files on a server, ensure that the files are assembled in deployable modules.

Next, refer to “Ways to update enterprise application files” on page 303 and decide how to update your application files. You can update enterprise applications or modules using the administrative console, the wsadmin tool, or Java MBean programming. These ways provide similar updating capabilities.

Further, ensure that the updated files can be installed to your deployment targets.

About this task

This topic describes how to update deployed applications or modules using the administrative console.

1. Back up the installed application or module.
 - a. Go to the Enterprise applications page of the administrative console. Click **Applications** → **Application Types** → **WebSphere enterprise applications** in the console navigation tree.
 - b. Export the application to an EAR file or export a file in the application. Select the application you want to export and click **Export** or **Export File**. Exporting preserves the binding information.
2. With the application selected on the Enterprise applications page, click **Update**. The Preparing for application update page is displayed.
3. Under **Specify the EAR, WAR, SAR or JAR module to upload and install**:
 - a. Ensure that **Application to be updated** refers to the application to be updated.
 - b. Under **Application update options**, select the installed application, module, or file that you want to update.

The online help Preparing for application update settings provides detailed information on the options.

Note: You cannot add, remove, or modify a Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) annotation using the **Replace or add a single file** or **Replace, add, or delete multiple files** update options. These options change a single file or a partial application. If you change a JAX-WS annotation using either of these options, the product does not return an error. However, you might encounter problems deploying annotated Web services.

4. If you selected the **Replace the entire application** or **Replace or add a single module** option:
 - a. Click **Next** to display a wizard for updating application files.
 - b. Complete the steps in the update wizard.

This update wizard, which is similar to the installation wizard, provides fields for specifying or editing application binding information. Refer to information on installing applications and on the application installation binding settings page for guidance.

Note that the installation steps have the merged binding information from the new version and the old version. If the new version has bindings for application artifacts such as Enterprise JavaBeans (EJB) Java Naming and Directory Interface (JNDI) names, EJB references or resource references, then those bindings will be part of the merged binding information. If new bindings are not present, then bindings are taken from the installed (old) version. If bindings are not present in the old version and if the default binding generation option is enabled, then the default bindings will be part of the merged binding information.

You can select whether to ignore bindings in the old version or ones in the new version.

5. Click **Finish**.
6. If you did not use the Manage modules page of the update wizard, after updating the application, map the installed application or module to servers.

Use the page accessed from the Enterprise Applications page.

- a. Go to the Manage modules page. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → *application_name* → **Manage modules**.
- b. Specify the application server where you want to install modules contained in your application and click **OK**.

You can deploy Java 2 Platform, Enterprise Edition (J2EE) 1.4 modules to servers on Version 6 or later nodes. You can deploy Java Platform, Enterprise Edition (Java EE) 5 modules to servers on Version 7.x nodes only.

Results

After replacement of a full application, the product uninstalls the old application. After replacement of a module, file or partial application, the product removes the old installed module, file or partial application from the installed application.

What to do next

After the application file or module installs successfully, do the following:

1. Save the changes to your configuration.
2. If needed, restart the application manually so the changes take effect.
If the application is updated while it is running, the product automatically stops the application or only its changed components, updates the application logic, and restarts the stopped application or its components.
3. If the application you are updating is deployed on a server that has its application class loader policy set to `Single` on the application server settings page, restart the server.

Preparing for application update settings

Use this page to update enterprise applications, modules or files already installed on a server.

To view this administrative console page, do the following:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select the installed application or module that you want to update.
3. Click **Update**.

Clicking **Update** displays a page that helps you update application files deployed in the cell. You can update the full application, a single module, a single file, or part of the application. If a new file or module has the same relative path as a file or module already existing on the server, the new file or module replaces the existing file or module. If the new file or module does not exist on the server, it is added to the deployed application.

Application to be updated

Specifies the name of the installed (or deployed) application that you selected on the Enterprise applications page.

Replace the entire application

Under **Application update options**, specifies to replace the application already installed on the server with a new (updated) enterprise application .ear file.

After selecting this option, do the following:

1. Specify whether the .ear file is on a local or remote file system and the full path name of the application. The path provides the location of the updated .ear file before installation.

Use **Local file system** if the browser and the updated files or modules are on the same machine, whether or not the server is on that machine too. **Local file system** is available for all update options. Use **Remote file system** if the application file resides on any node in the current cell context. Also use the **Remote file system** option to specify an application file already residing on the machine running the application server. For example, the field value might be `app_server_install_root/installableApps/test.ear`. If you are installing a standalone WAR module, then specify the context root as well.

Note: During application installation, application files typically are uploaded from a client machine running the browser to the server machine running the administrative console, where they are deployed. In such cases, use the Web browser running the administrative console to select modules to upload to the server machine. In some cases, however, the application files reside on the file system of any of the nodes in a cell. To have the application server install these files, use the **Remote file system** option.

2. If you are installing a standalone Web application (WAR) or a Session Initiation Protocol (SIP) module (SAR), specify the context root of the WAR or SAR file.
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. For example, if the context root is `/gettingstarted` and the servlet mapping is `MySession`, then the URL is `http://host:port/gettingstarted/MySession`.
3. Click **Next** to display a wizard for updating application files. The update wizard, which is similar to the installation wizard, provides fields for specifying or editing application binding information. Complete the steps in the update wizard as needed.

When the full application is updated, the old application is uninstalled and the new application is installed. When the configuration changes are saved and subsequently synchronized, the application files are expanded on the node where application will run. If the application is running on the node while it is updated, then the application is stopped, application files are updated, and application is started.

Replace or add a single module

Under **Application update options**, specifies to replace a module in or add a module to an installed application.

The module can be a Web module (.war file), enterprise bean module (EJB .jar file), SIP module (.sar file), or resource adapter module (connector .rar file).

After selecting this option, specify whether the module is on a local or remote file system and the full path name of the module. The path provides the location of the updated module before installation. For information on **Local file system** and **Remote file system**, refer to the previous description of **Replace the entire application**.

To replace a module, the specified relative path (module URI) must match the path of the module to be updated in the installed application.

To add a new module to the installed application, the specified relative path must *not* match the path of a module in the installed application. The value specifies the desired path for the new module.

If you are installing a standalone Web or SIP module, specify a value for **Context root**. 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. For example, if the context root is `/gettingstarted` and the servlet mapping is `MySession`, then the URL is `http://host:port/gettingstarted/MySession`.

Next, specify whether to show only installation options that require you to supply information or to show all installation options.

After specifying the required information on the module, click **Next** to display a wizard for updating application files. The update wizard, which is similar to the installation wizard, provides fields for specifying or editing module binding information. Complete the steps in the update wizard as needed.

After a single module is added or updated, when configuration changes are saved, the new or updated module is stored in the deployed application in the product configuration repository. When these changes are synchronized with the node, the module is added or updated to the node's file system. If the application is running on the node when the module is added or updated, then one of the following occurs:

- For updates to a Web module, the running Web module is stopped, Web module files are updated, and then the Web module is started.
- For module additions, the added module is started on the application servers where the application is running after it is expanded on the node. An application restart is not necessary.
- If the class loader policy for the application is set to `Single` so that all modules share a class loader, then the entire application is stopped and restarted for module level changes.
- If the security provider configured with the product does not support dynamic updates, then the entire application is stopped and restarted for module level changes.
- For all other updates to a module, the entire application is stopped, the module files are updated, then the entire application is started.

Replace or add a single file

Under **Application update options**, specifies to replace a file in or add a file to an installed application.

Use this option to update a file used by the application that is not an `.ear`, `.war`, `.sar`, `.rar` or, in some instances, a `.jar` file. You can use this option to add or update `.jar` files that are not defined as modules in the application. To update an `.ear` file use the **Replace the entire application** option. To update a `.war` file, `.sar` file, `.rar` file, or `.jar` file that is defined as a module in the application, use the **Replace or add a single module** option.

After selecting this option, specify whether the file is on a local or remote file system and the full path name of the file. The path provides the location of the updated file before installation. For information on **Local file system** and **Remote file system**, refer to the description of **Replace the entire application**.

For the relative path, specify a relative path to the file that starts from the root of the `.ear` file. For example, if the file is located at `com/company/greeting.class` in module `hello.jar`, specify a relative path of `hello.jar/com/company/greeting.class`.

To replace a file, the relative path must match the path of the file to be updated in the installed application.

To add a new file to the installed application, the must *not* match the path of a file in the installed application. The value specifies the desired path for the new file.

After you specify the file system and relative paths, click **Next**.

After a single file is added or updated, when configuration changes are saved, the new or updated file is stored in the deployed application in the product configuration repository. When these changes are synchronized with the node, the file is added or updated to the node's file system. If the application is running on the node when the file is added or updated, then one of the following occurs:

- When files are added at application metadata scope (META-INF directory) or updated at any application scope or in non-Web modules, the entire application is stopped, the file is added or updated, and then the entire application is restarted.
- When files are added at application non-metadata scope (outside of META-INF directory but not in any module), the changes are saved in the file system without restarting the running application.
- When files are added or updated to Web module metadata (META-INF or WEB-INF directory), the running Web module is stopped, the Web module file is added or updated, and then the Web module is started.

- For all other files in Web modules, the file is added or updated on the node's file system without stopping the application or any of its components.

Replace, add, or delete multiple files

Under **Application update options**, specifies to update multiple files of an installed application by uploading a compressed file. Depending on the contents of the compressed file, a single use of this option can replace files in, add new files to, and delete files from the installed application. Each entry in the compressed file is treated as a single file and the path of the file from the root of the compressed file is treated as the relative path of the file in the installed application.

After selecting this option, specify whether the compressed file is on a local or remote file system and the full path name of the compressed file. You will likely use **Local file system** because you are uploading a compressed file and remote browsing only works for .ear, .sar, .war or .jar files. Specify a valid compressed file format such as .zip or .gzip. The path provides the location of the compressed file before installation. This option unzips the compressed file into the installed application directory.

Use **Local file system** if the browser and the updated files or modules are on the same machine, whether or not the server is on that machine too. **Local file system** is available for all update options.

To replace a file, a file in the compressed file must have the same relative path as the file to be updated in the installed application.

To add a new file to the installed application, a file in the compressed file must have a different relative path than the files in the installed application.

The relative path of a file in the installed application is formed by concatenation of the relative path of the module (if the file is inside a module) and the relative path of the file from the root of the module separated by /.

To remove a file from the installed application, specify metadata in the compressed file using a file named META-INF/ibm-partialapp-delete.props at any archive scope. The ibm-partialapp-delete.props file must be an ASCII file that lists files to be deleted in that archive with one entry for each line. The entry can contain a string pattern such as a regular expression that identifies multiple files. The file paths for the files to be deleted must be relative to the archive path that has the META-INF/ibm-partialapp-delete.props file.

Level of files to delete	Metadata .props file to include in compressed file
Application	<p>Include META-INF/ibm-partialapp-delete.props in the compressed file. In the metadata .props file, list files to be deleted. File paths are relative to the location of the META-INF/ibm-partialapp-delete.props file.</p> <p>For example, to delete a file named utils/config.xml from the root of the my.ear file, include the line utils/config.xml in the META-INF/ibm-partialapp-delete.props file.</p>
Module	<p>Include <i>module_uri</i>/META-INF/ibm-partialapp-delete.props in the compressed file.</p> <p>To delete one file from a module, include the file path relative to the module in the metadata .props file. For example, to delete a/b/c.jsp from the my.jar module, include a/b/c.jsp in my.jar/META-INF/ibm-partialapp-delete.props file in the compressed file.</p> <p>To delete multiple files within a module, list the files to be deleted in the metadata .props file with one entry on each line. For example, to delete all JavaServer Pages (.jsp files) from the my.war file, include the line *.jsp in the my.war/META-INF/ibm-partialapp-delete.props file. The line uses a regular expression, *.jsp, to identify all .jsp files in my.war.</p>

You can use a single partial application file to add, delete and update multiple files.

After you specify a file system path, click **Next**.

After a partial application update, when configuration changes are saved, the new or updated application file is stored in the deployed application in the WebSphere Application Server configuration repository. When these changes are synchronized with the node, the files are added or updated to the node's file system. Because the partial application option updates multiple files, the application components that are restarted are determined using individual files in the partial application.

An example of entries in a partial application compressed file follows:

```
util.jar
META-INF/ibm-partialapp-delete.props
foo.jar/com/mycomp/xyz.class
xyz.war/welcome.jsp
xyz.war/WEB-INF/web.xml
webmod.war/META-INF/ibm-partialapp-delete.props
```

For this example, the META-INF/ibm-partialapp-delete.props file contains the *.dat and tools/test.jar files. The webmod.war/META-INF/ibm-partialapp-delete.props file contains the com/test/*.jsp and WEB-INF/test.xmi files.

The partial application update option does the following:

- Adds or replaces util.jar in the deployed application.
- Adds or replaces com/mycomp/xyz.class inside the foo.jar file of the deployed application.
- Deletes *.dat files from the application, but not from any modules.
- Deletes tools/test.jar from the application.
- Adds or replaces welcome.jsp inside the xyz.war module of the deployed application.
- Replaces WEB-INF/web.xml inside the xyz.war module of the deployed application.
- Deletes com/test/*.jsp from the webmod.war module.
- Deletes WEB-INF/test.xmi from the webmod.war module.

Hot deployment and dynamic reloading

You can make various changes to applications and their modules without having to stop the server and start it again. Making these types of changes is known as *hot deployment and dynamic reloading*.

Before you begin

This topic assumes that your application files are deployed on a server and you want to upgrade the files.

See “Ways to update enterprise application files” on page 303 and determine whether hot deployment is the appropriate way for you to update your application files. Other ways are easier and hot deployment is appropriate only for experienced users.

Do not use hot deployment if you intend to export your application, generate a plug-in based on the application configuration, or perform other application management in the future. Changes that you make to your application files using hot deployment are not recognized by administrative console or wsadmin application management functions. Those functions recognize only the application files that administrative programs such as the console or wsadmin present during application installation, update or other management functions. The application management functions do not recognize files changed by hot deployment.

About this task

Hot deployment is the process of adding new components (such as WAR files, EJB Jar files, enterprise Java beans, servlets, and JSP files) to a running server without having to stop the application server process and start it again.

Dynamic reloading is the ability to change an existing component without needing to restart the server in order for the change to take effect. Dynamic reloading involves:

- Changes to the implementation of a component of an application, such as changing the implementation of a servlet
- Changes to the settings of the application, such as changing the deployment descriptor for a Web module

As opposed to the changes made to a deployed application described in “Updating enterprise application files” on page 302, changes made using hot deployment or dynamic reloading do not use the administrative console or a wsadmin scripting command. You must directly manipulate the application files on the server where the application is deployed.

If the application you are updating is deployed on a server that has its application class loader policy set to Single, you might not be able to dynamically reload your application. At minimum, you must restart the server after updating your application.

1. Locate your expanded application files.

The application files are in the directory you specified when installing the application or, if you did not specify a custom target directory, are in the default target directory, *app_server_root/installedApps/cell_name*. Your EAR file, */\${APP_INSTALL_ROOT}/cell_name/application_name.ear*, points to the target directory. The *variables.xml* file for the node defines */\${APP_INSTALL_ROOT}*.

It is important to locate the expanded application files because, as part of installing applications, a WebSphere application server unjars portions of the EAR file onto the file system of the computer that will run the application. These expanded files are what the server looks at when running your application. If you cannot locate the expanded application files, look at the *binariesURL* attribute in the *deployment.xml* file for your application. The attribute designates the location the run time uses to find the application files.

For the remainder of this information on hot deployment and dynamic reloading, *application_root* represents the root directory of the expanded application files.

2. Locate application metadata files. The metadata files include the deployment descriptors (*web.xml*, *application.xml*, *ejb-jar.xml*, and the like), the bindings files (*ibm-web-bnd.xmi*, *ibm-app-bnd.xmi*, and the like), and the extensions files (*ibm-web-ext.xmi*, *ibm-app-ext.xmi*, and the like).

Metadata XML files for an application can be loaded from one of two locations. The metadata files can be loaded from the same location as the application binary files (such as *application_root/META-INF*) or they can be loaded from the WebSphere configuration tree, */\${CONFIG_ROOT}/cells/cell_name/applications/application_EAR_name/deployments/application_name/*. The value of the *useMetadataFromBinary* flag specified during application installation controls which location is used. If specified, the metadata files are loaded from the same location as the application binary files. If not specified, the metadata files are loaded from the application deployment folder in the configuration tree.

Note: You can have *useMetadataFromBinaries=true*, change an extracted copy of your application using hot deployment, and have the changes take effect at run time by following the procedure in this topic. However, changes that you make to your application files using hot deployment are not recognized by console or wsadmin application management functions. Those functions recognize only the original application files and not the files changed by hot deployment. Do not use hot deployment if you intend to export your application, generate a plug-in based on the application configuration, or perform other application management in the future. Hot deployment enables you to quickly change application files; it does not support the full management lifecycle of an application.

For the remainder of this information, *metadata_root* represents the location of the metadata files for the specified application or module.

3. Optional: Examine the values specified for **Reload classes when application files are updated** and **Polling interval for updated files** on the settings page for your application’s class loader.

If reloading of classes is enabled and the polling interval is greater than zero (0), the application files are reloaded after the application is updated. For JavaServer Pages (JSP) files in a Web module, a Web container reloads JSP files only when the IBM extension `jspReloadingEnabled` in the `jspAttributes` of the `ibm-web-ext.xmi` file is set to `true`. You can set `jspReloadingEnabled` to `true` when editing your Web module's extended deployment descriptors in an assembly tool.

4. Change or add the following components or modules as needed:
 - Application files
 - WAR files
 - EJB Jar files
 - HTTP plug-in configuration files
5. For changes to take effect, you might need to start, stop, or restart an application.

“Starting or stopping enterprise applications” on page 296 provides information on using the administrative console to start, stop, or restart an application.

Results

The application files are updated on the server.

Because you directly manipulated the application files on the server, you might not be able to later use the administrative console or a `wsadmin` scripting command to work with the files. For example, if you try exporting a manually changed application using **Export** on an Enterprise applications console page, your manual changes to an application in the `installedApps` directory are not exported. To export those changes, you must copy and move the application files manually.

Changing or adding application files

You can change or add application files on application servers without having to stop the server and start it again.

About this task

There are several changes that you can make to deployed application files without stopping the server and starting it again.

Note: See “Ways to update enterprise application files” on page 303 and determine whether hot deployment is the appropriate way for you to update your application files. Other ways are easier and hot deployment is appropriate only for experienced users. You can use the update wizard of the administrative console to make the changes without having to stop and restart the server.

The following table lists the changes that you can make by manipulating an application file on the server where the application is deployed. The table also states whether you use hot deployment or dynamic reloading to make the changes.

Change	Hot deployment	Dynamic reloading
Update an existing application on a running server by providing a new EAR file.	Yes	Yes
Add a new application to a running server.	Yes	No
Remove an existing application from a running server.	Yes	No
Change or add files to existing EJB or Web modules.	Yes	No
Change the <code>application.xml</code> file for an application.	Not applicable	Yes
Change the <code>ibm-app-ext.xmi</code> file for an application.	Not applicable	Yes
Change the <code>ibm-app-bnd.xmi</code> file for an application.	Not applicable	Yes
Change a non-module Jar file contained in the EAR file.	Yes	Yes

- Update an existing application on a running server by providing a new EAR file.
Reinstall an updated application using the administrative console or the `wsadmin $AdminApp install` command with the `-update` option.
Both reinstallation methods enable you to update an existing application using any of the other steps listed in this file, including changing classes, adding modules, removing modules, changing modules, or changing metadata files. The application reinstallation methods detect the changes in your application and prompt you for additional binding data that might be needed to install the application. The reinstallation process automatically stops and restarts your application on the appropriate servers.
- Add a new application to a running server.
Install an application using the administrative console or the `wsadmin install` command.
- Remove an existing application from a running server.
Stop the application and then uninstall it from the server. Use the administrative console to stop the application and then uninstall it. Or use the `stopApplication` attribute of the `AdminControl` object with the `wsadmin` tool and then run the `uninstall` command.
- Change or add files to existing EJB or Web modules.
 1. Update the application files in the *application_root* location.
 2. Restart the application.
Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
- Change the `application.xml` file for an application.
Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
- Change the `ibm-app-ext.xmi` file for an application.
Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
- Change the `ibm-app-bnd.xmi` file for an application.
Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
- Change a non-module Jar file contained in the EAR file.
 1. Update the non-module Jar file in the *application_root* location.
 2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

Changing or adding WAR files

You can change Web application archives (WAR files) on application servers without having to stop the server and start it again.

About this task

There are several changes that you can make to WAR files without stopping the server and starting it again.

Note: See “Ways to update enterprise application files” on page 303 and determine whether hot deployment is the appropriate way for you to update your WAR files. Other ways are easier and hot

deployment is appropriate only for experienced users. You can use the update wizard of the administrative console to make the changes without having to stop and restart the server.

The following table lists the changes that you can make by manipulating a WAR file on the server where the application is deployed. The table also states whether you use hot deployment or dynamic reloading to make the changes.

Change	Hot deployment	Dynamic reloading
Change an existing JavaServer Pages (JSP) file.	Not applicable	Yes
Add a new JSP file to an existing application.	Yes	Yes
Change an existing servlet class by editing and recompiling.	Not applicable	Yes
Change a dependent class of an existing servlet class.	Not applicable	Yes
Add a new servlet using the Invoker (Serve Servlets by class name) facility or add a dependent class to an existing application.	Yes	Not applicable
Add a new servlet, including a new definition of the servlet in the web.xml deployment descriptor for the application.	Yes	Not applicable
Change the web.xml file of a WAR file.	Yes	Yes
Change the ibm-web-ext.xmi file of a WAR file.	Not applicable	Yes
Change the ibm-web-bnd.xmi file of a WAR file.	Not applicable	Yes

- Change an existing JavaServer Pages (JSP) file.
Place the changed JSP file directly in the *application_root/module_name* directory or the appropriate subdirectory. The change will be automatically detected and the JSP will be recompiled and reloaded.
- Add a new JSP file to an existing application.
Place the new JSP file directly in the *application_root/module_name* directory or the appropriate subdirectory. The new file will be automatically detected and compiled on the first request to the page.
- Change an existing servlet class by editing and recompiling.
 1. Place the new version of the servlet .class file directly in the *application_root/module_name/WEB-INF/classes* directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in *application_root/module_name/WEB-INF/lib*. In either case, the change will be detected, the Web application will be shut down and reinitialized, picking up the new class.
 2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.
- Change a dependent class of an existing servlet class.
 1. Place the new version of the dependent .class file directly in the *application_root/module_name/WEB-INF/classes* directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in *application_root/module_name/WEB-INF/lib*. In either case, the change will be detected, the Web application will be shut down and reinitialized, picking up the new class.
 2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.
- Add a new servlet using the Invoker (Serve Servlets by class name) facility or add a dependent class to an existing application.
 1. Place the new .class file directly in the *application_root/module_name/WEB-INF/classes* directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in

application_root/module_name/WEB-INF/lib. In either case, the change will be detected, the Web application will be shut down and reinitialized, picking up the new class.

This case is treated the same as changing an existing class. The difference is that adding the servlet or class does not immediately cause the Web application to reload because the class has never been loaded before. The class simply becomes available for execution.

2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.

If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

- Add a new servlet, including a new definition of the servlet in the `web.xml` deployment descriptor for the application.
 1. Place the new `.class` file directly in the *application_root/module_name/WEB-INF/classes* directory. If the `.class` file is part of a Jar file, you can place the new version of the Jar file directly in *application_root/module_name/WEB-INF/lib*.

You can edit the `web.xml` file in place or copy it into the *application_root/module_name/WEB-INF/classes* directory. The new `.class` file will not trigger a reloading of the application.
 2. Restart the application.

Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool. After the application restarts, the new servlet is available for service.
- Change the `web.xml` file of a WAR file.
 1. Edit the `web.xml` file in place or copy it into the *metadata_root/module_name/WEB-INF* directory.
 2. Restart the application.

Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
- Change the `ibm-web-ext.xmi` file of a WAR file.

Edit the extension settings as needed. You can change all of the extension settings. The only warning is if you set the `reloadInterval` property to zero (0) or the `reloadEnabled` property to `false`, the application no longer automatically detects changes to class files. Both of these changes disable the automatic reloading function. The only way to re-enable automatic reloading is to change the appropriate property and restart the application. See other task descriptions in this file for information on restarting an application.
- Change the `ibm-web-bnd.xmi` file of a WAR file.
 1. Edit the bindings as needed. You can change all of the values but ensure that the entities you are binding to are present in the configuration of the server.
 2. Restart the application.

Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.

Changing or adding EJB Jar files

You can change enterprise bean (EJB) Jar files on application servers without having to stop the server and start it again.

About this task

There are several changes that you can make to EJB Jar files without stopping the server and starting it again.

Note: See “Ways to update enterprise application files” on page 303 and determine whether hot deployment is the appropriate way for you to update your EJB Jar files. Other ways are easier and

hot deployment is appropriate only for experienced users. You can use the update wizard of the administrative console to make the changes without having to stop and restart the server.

The following table lists the changes that you can make to EJB Jar files by manipulating an EJB file on the server where the application is deployed. The table also states whether you use hot deployment or dynamic reloading to make the changes.

Change	Hot deployment	Dynamic reloading
Change the <code>ejb-jar.xml</code> file of an EJB Jar file.	Not applicable	Yes
Change the <code>ibm-ejb-jar-ext.xmi</code> or <code>ibm-ejb-jar-bnd.xmi</code> file of an EJB Jar file.	Not applicable	Yes
Change the <code>Table.ddl</code> file for an EJB Jar file.	Not applicable	Not applicable
Change the <code>Map.mapxmi</code> or <code>Schema.dbxmi</code> file for an EJB Jar file.	Not applicable	Yes
Update the implementation class for an EJB file or a dependent class of the implementation class for an EJB file.	Not applicable	Yes
Update the Home/Remote interface class for an EJB file.	Not applicable	Yes
Add a new EJB file to an existing EJB Jar file.	Yes	Yes

- Change the `ejb-jar.xml` file of an EJB Jar file.
Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
- Change the `ibm-ejb-jar-ext.xmi` or `ibm-ejb-jar-bnd.xmi` file of an EJB Jar file.
Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
- Change the `Table.ddl` file for an EJB Jar file.
Rerun the DDL file on the user database server. Changing the `Table.ddl` file has no effect on the application server and is a change to the database table schema for the EJB files.
- Change the `Map.mapxmi` or `Schema.dbxmi` file for an EJB Jar file.
 1. Change the `Map.mapxmi` or `Schema.dbxmi` file for an EJB Jar file.
 2. Regenerate the deployed code artifacts for the EJB file.
 3. Apply the new EJB Jar file to the server.
 4. Restart the application. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
- Update the implementation class for an EJB file or a dependent class of the implementation class for an EJB file.
 1. Update the class file in the `application_root/module_name.jar` file.
 2. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.
If automatic reloading is not enabled, restart the application of which the EJB file is a member. If the updated module is used by other modules in other applications, restart those applications as well. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.
- Update the Home/Remote interface class for an EJB file.
 1. Update the interface class of the EJB file.
 2. Regenerate the deployed code artifacts for the EJB file.
 3. Apply the new EJB Jar file to the server.

4. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

If automatic reloading is not enabled, restart the application of which the EJB file is a member. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.

- Add a new EJB file to an existing EJB Jar file.

1. Apply the new or updated Jar file to the `application_root` location.
2. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or use the `startApplication` and `stopApplication` attributes of the `AdminControl` object with the `wsadmin` tool.

Changing the HTTP plug-in configuration

You can change the HTTP plug-in configuration without having to stop the server and start it again.

About this task

There are several change that you can make to the HTTP plug-in configuration without stopping the server and starting it again.

Note: See “Ways to update enterprise application files” on page 303 and determine whether hot deployment is the appropriate way for you to update your HTTP plug-in configuration. Other ways are easier and hot deployment is appropriate only for experienced users.

The following table lists the changes that you can make to the HTTP plug-in configuration. The table also states whether you use hot deployment or dynamic reloading to make the changes.

Change	Hot deployment	Dynamic reloading
Change the <code>application.xml</code> file to change the context root of a Web application archive (WAR file).	Yes	No
Change the <code>web.xml</code> file to add, remove, or modify a servlet mapping.	Yes	Yes
Change the <code>server.xml</code> file to add, remove, or modify an HTTP transport or change the <code>virtualhost.xml</code> file to add or remove a virtual host or to add, remove, or modify a virtual host alias.	Yes	Yes

- Change the `application.xml` file to change the context root of a WAR file.
 1. Change the `application.xml` file.
 2. If the plug-in configuration property **Automatically propagate plug-in configuration file** is selected for this plug-in, it is automatically regenerated whenever the `application.xml` file changes.

See documentation on the Web server plug-in properties for information on how to set this property. You can also run the `GenPluginCfg.bat/sh` script, or issue a `wsadmin` command to regenerate the plug-in configuration file.
- Change the `web.xml` file to add, remove, or modify a servlet mapping.
 1. Change the `web.xml` file.
 2. If the plug-in configuration property **Automatically propagate plug-in configuration file** is selected for this plug-in, it is automatically regenerated whenever the `web.xml` file changes.

See documentation on the Web server plug-in properties for information on how to set this property. You can also run the `GenPluginCfg.bat/sh` script, or issue a `wsadmin` command to regenerate the plug-in configuration file.

If the Web application has file serving enabled or has a servlet mapping of `/`, the plug-in configuration does not have to be regenerated. In all other cases a regeneration is required.

- Change the `server.xml` file to add, remove, or modify an HTTP transport or change the `virtualhost.xml` file to add or remove a virtual host or to add, remove, or modify a virtual host alias.
 1. Change the `server.xml` file or the `virtualhost.xml` file.
 2. If the plug-in configuration property **Automatically propagate plug-in configuration file** is selected for this plug-in, it is automatically regenerated whenever the `server.xml` file changes.
See documentation on the Web server plug-in properties for information on how to set this property. You can also run the `GenPluginCfg.bat/sh` script, or issue a `wsadmin` command to regenerate the plug-in configuration file.

Uninstalling enterprise applications

After an application no longer is needed, you can uninstall it.

About this task

Uninstalling an application deletes the application from the product configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** in the administrative console navigation tree to access the Enterprise applications page.
2. If you need to retain a copy of the application, back up the application.
 - a. Select the application to uninstall.
 - b. Click **Export**.

The product exports the application to an enterprise application (.ear) file, preserving the binding information.

3. Uninstall the application.
 - a. Select the application to uninstall.
 - b. Click **Uninstall**.
 - c. On the Uninstall application panel, click **OK**.
4. Save changes made to the administrative configuration.

Results

On single-server products, application binaries are deleted after you save the changes.

Removing enterprise files

After a file is no longer needed, you can remove the file from a Java Platform, Enterprise Edition (Java EE) application or module deployed on a server.

About this task

Removing a file deletes the file from the product configuration repository and deletes the file from the file system of all nodes where the file is installed.

You can use the administrative console to remove a file from an application or module.

- Remove a file from an application.
 1. Go to the Enterprise applications page. Click **Applications** → **Application Types** → **WebSphere enterprise applications** in the console navigation tree.
 2. Select the application that contains a file you want removed.
 3. Click **Remove File**. The Remove a file page is displayed.
 4. Select the URI of the file that you want removed from the application.

5. Back up the application.
Under **Export before removing file**, select the application name and then specify the location to which you want the file exported.
 6. Click **OK** to remove the file.
- Remove a file from a module.
 1. Go to the Manage modules page.
Click **Applications** → **Application Types** → **WebSphere enterprise applications** → **application_name** → **Manage modules** in the console navigation tree.
 2. Select the module from which you want to delete a file.
 3. Click **Remove File**. The Remove a file from a module page is displayed.
 4. Select the URI of the file that you want removed from the module.
 5. Back up the application.
Under **Export before removing file**, select the application name and then specify the location to which you want the file exported.
 6. Click **OK** to remove the file.

Results

The file is exported to the designated location and removed from the application or module. The application or standalone Web module that had a file removed is restarted so the changes take effect.

What to do next

Save the changes to your administrative configuration.

On single-server products, application binaries are deleted after you save the changes.

Deploying and administering applications: Resources for learning

Use the following links to find relevant supplemental information about deploying and administering applications using the administrative console. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- “Programming model and decisions”
- “Programming instructions and examples” on page 320
- “Administration” on page 320

Programming model and decisions

- Designing Enterprise Applications with the Java™ 2 Platform, Enterprise Edition, Second Edition, http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/
- Java EE Tutorials, <http://java.sun.com/javaee/reference/tutorials/>
- Recommended reading list: J2EE and WebSphere Application Server, http://www.ibm.com/developerworks/websphere/library/techarticles/0305_issw/recommendedreading.html
- Java EE 5: Power and productivity with less complexity – An overview of Java EE 5 features and developer-productivity enhancements, <http://www.ibm.com/developerworks/java/library/j-jee5/index.html?ca=drs->

- Rational Application Developer V7 Programming Guide, SG24-7501-00, <http://www.redbooks.ibm.com/abstracts/sg247501.html?Open>
- *IBM WebSphere Developer Technical Journal: The top Java EE best practices*, http://www.ibm.com/developerworks/websphere/techjournal/0701_botzum/0701_botzum.html

Programming instructions and examples

- *IBM WebSphere: Deployment and Advanced Configuration*, Roland Barcia, *et al.*, ISBN 0131468626 (Prentice Hall, 2004)
- *IBM WebSphere Developer Technical Journal: Co-hosting multiple versions of J2EE applications*, http://www.ibm.com/developerworks/websphere/techjournal/0405_poddar/0405_poddar.html
- Automated Deployment of Enterprise Application Updates: Part 1 - Basic concepts, <http://websphere.sys-con.com/read/47889.htm>

Administration

- *IBM WebSphere Developer Technical Journal: System management for WebSphere Application Server V6 -- Part 1 Overview of system management enhancements*, http://www.ibm.com/developerworks/websphere/techjournal/0501_williamson/0501_williamson.html
- *IBM WebSphere Developer Technical Journal: System management for WebSphere Application Server V6 -- Part 5: Flexible options for updating deployed applications*, http://www.ibm.com/developerworks/websphere/techjournal/0510_apte/0510_apte.html
- *WebSphere Application Server V6.1: System Management Configuration Handbook*, SG24-7304-00, <http://www.redbooks.ibm.com/abstracts/SG247304.html?Open>

Chapter 8. Managing applications through programming

Through Java MBean programming, you can install, update, and delete a Java Platform, Enterprise Edition (Java EE) application on a WebSphere Application Server deployment target.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming, see MBean Java application programming interface (API) documentation.

For information on the restarting of updated applications, refer to Fine-grained recycle behavior in *IBM WebSphere Developer Technical Journal: System management for WebSphere Application Server V6 -- Part 5 Flexible options for updating deployed applications*.

Before you can install or change an application on a deployment target, you must first create or update your application and assemble it using an assembly tool.

About this task

Besides installing, uninstalling, and updating applications through programming, you can additionally install, uninstall, and update Java EE applications through the administrative console or the wsadmin tool. All three ways provide identical updating capabilities.

1. Perform any or all of the following tasks to manage your Java EE applications through programming.
 - Access the application management function.

This topic provides examples to access the application management functionality:

 - From WebSphere Application Server code
 - From outside WebSphere Application Server
 - When WebSphere Application Server is not running
 - Install an application.

This topic provides an example for initially installing an application on a deployment target such as a server .
 - Uninstall an application.

This topic provides an example for uninstalling an application that resides on a deployment target.
 - Manipulate additional attributes for a deployed application.

This topic provides an example for manipulating attributes that are not exposed through the `AppDeploymentTask` object.
 - Share sessions for application management.

This topic provides an example for saving application-specific updates for a deployed application to a session, and then to the configuration repository.
 - Update an application.

This topic provides an example for updating the installed application on a server with a new application. When you completely update an application, the deployed application is uninstalled and the new enterprise archive (EAR) file is installed.
 - Add to, update, or delete part of an application.

This topic provides an example that you can use to add, update, or delete part of an application on a server .
 - Edit an application.

This topic provides an example that you can use to edit an application on a server .
 - Add a module.

This topic provides an example for adding a module to an application that resides on a server .

- Update a module.

This topic provides an example for updating a module that resides on a server . When you update a module, the deployed module is uninstalled and the updated module is installed.

- Delete a module.

This topic provides an example for deleting a module that resides on a server . When you delete a module, the deployed module is uninstalled.

- Add a file.

This topic provides an example for adding a file to an application that resides on a server .

- Update a file.

This topic provides an example for updating a file on a server . When you update a file, the deployed file is uninstalled and the updated file is installed.

- Delete a file.

This topic provides an example for deleting a file on a server . When you delete a file, the deployed file is uninstalled.

2. Save your changes to the master configuration repository.

What to do next

If you have further application updates, you can do the updates through programming, the administrative console, or the wsadmin tool.

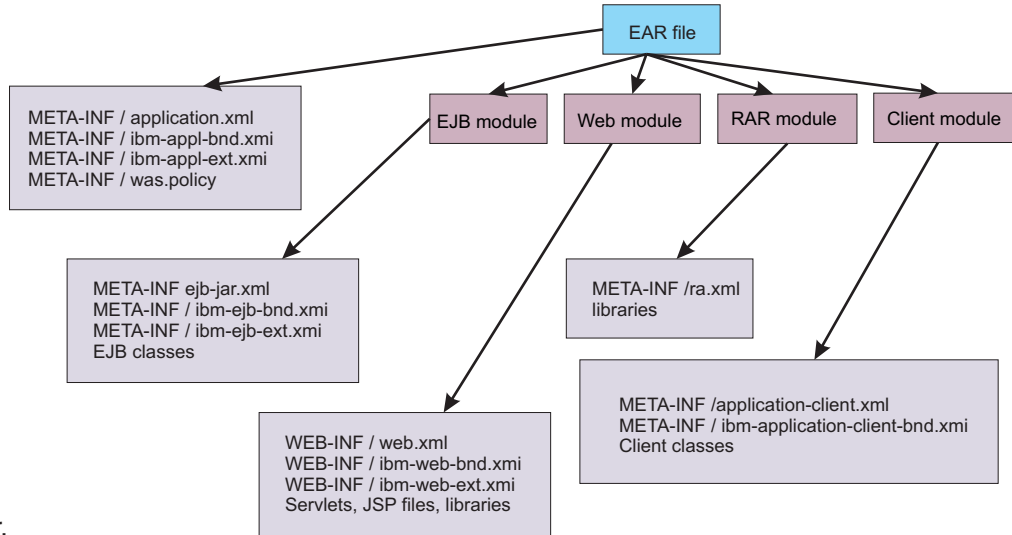
You can use the common deployment framework to add additional logic to application management operations. See “Extending application management operations through programming” on page 355. The tasks that the extensions provide are available through all the administrative clients, such as the wsadmin tool, the administrative console, or through programmatic APIs that the AppManagement MBean provides.

Application management

Java 2 Platform, Enterprise Edition (J2EE) applications and modules include an Extensible Markup Language (XML)-based deployment descriptor that specifies various J2EE artifacts that pertain to applications or modules. The J2EE artifacts include Enterprise JavaBeans (EJB) definitions, security role definitions, EJB references, resource references, and so on. These artifacts define various unresolved references that the application logic uses. The J2EE specification requires that these artifacts map to J2EE platform-specific information, such as that found in WebSphere Application Server, during deployment of J2EE applications.

The application assembly tools that WebSphere Application Server supports, as well as the application management support that is provided with the product, facilitate collection of certain WebSphere Application Server information. The collected information is used to resolve references that are defined in various deployment descriptors in a J2EE application. This information is stored in the application EAR file in conjunction with the deployment descriptors. The following diagram shows the structure of an Enterprise

Archive (EAR) file that is populated with deployment information that is specific to WebSphere Application



Server.

The application management architecture provides a set of classes with which deployers can collect WebSphere Application Server deployment information. This information is also referred to as *binding information*, and is stored in the application EAR file. The deployer can install the EAR file into a WebSphere Application Server configuration by using the AppManagement interface.

The application management support in WebSphere Application Server provides functions such as installing and uninstalling applications, editing binding information for installed applications, updating the entire application or part of the application, exporting the application, and so on. The `com.ibm.websphere.management.application.AppManagement` interface, which is exposed as a Java Management Extensions (JMX)-based AppManagement MBean in WebSphere Application Server, provides this functionality. Code that runs on the server or in a stand-alone administrative client program can access the interface. Access to the application management functions is also possible in the absence of WebSphere Application Server. This mode, known as *local mode*, is particularly useful for installing J2EE applications as part of product installation.

Accessing the application management function

The `com.ibm.websphere.management.application.AppManagementProxy` class provides uniform access to application management functionality, regardless of whether the functionality is accessed from the server process, administrative client process, or a stand-alone Java program in the absence of WebSphere Application Server. This topic provides code excerpts that demonstrate how to obtain an `AppManagementProxy` instance in a variety of cases.

Before you begin

This task assumes a basic familiarity with WebSphere Application Server programming interfaces and MBean programming. Read about WebSphere Application Server programming interfaces and MBean programming in the application programming interfaces documentation.

About this task

Perform any of the following tasks to access application management functionality through programming.

- To access application management functionality from WebSphere Application Server code, for example, as a custom service, create the `AppManagementProxy` class.

```
AppManagement appMgmt =
    AppManagementProxy.getJMXProxyForServer();
```

- To access application management functionality from outside WebSphere Application Server through the AppManagement MBean, create an administrative client to establish a connection to WebSphere Application Server and then create the AppManagementProxy class.

```
AdminClient adminClient = ....
```

```
// create AppManagement proxy object
AppManagement appMgmt = AppManagementProxy.getJMXProxyForClient(adminClient);
```

- To access application management functionality when WebSphere Application Server is not running (local mode), create the AppManagementProxy class.

```
AppManagement appMgmt = AppManagementProxy.getLocalProxy();
```

- When running in local mode set the `com.ibm.ws.management.standalone` system property to `true`. If you want to modify configuration documents in a non-default location, set the location of the configuration directory through the `was.repository.root` system property.
- Although you can use application management functions with or without WebSphere Application Server running, do not access application management functions concurrently through local mode and the AppManagement MBean. Otherwise, updates that are made using these modes can collide and break the integrity of the WebSphere Application Server configuration.

Results

After you successfully create the AppManagementProxy class, you have access to application management functionality.

What to do next

You can perform various management tasks such as installing, uninstalling, editing, and so on.

Installing an application through programming

You can install an application through the administrative console, the wsadmin tool, or programming. Use this example to install an application through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. Read about MBean programming in the application programming interfaces documentation.

Before you can install an application on WebSphere Application Server, you must first create or update your application and assemble it using an assembly tool.

About this task

The AppDeploymentController instance contains meta-data defined in XMLI-based deployment descriptors as well as annotations defined in Java classes within the input enterprise archive (EAR) file.

Perform the following tasks to install an application through programming.

1. Populate the EAR file with WebSphere Application Server-specific binding information.
 - a. Create the controller and populate the EAR file with appropriate options.
 - b. Optionally run the default binding generator.
 - c. Save and close the EAR file.
 - d. Retrieve the saved options table that will be passed to the installApplication MBean API.
2. Connect to WebSphere Application Server.
3. Create the application management proxy.

4. If the preparation phase (population of the EAR file) is not performed, the do the following actions:
 - a. Create an options table to be passed to the installApplication MBean API.
 - b. Create a table for module to server relations and add the table to the options table.
 Refer to the `com.ibm.websphere.management.application.AppManagement` class in the application programming interfaces documentation to understand various options that can be passed to the installApplication MBean API.
5. Create the notification filter for listening to installation events.
6. Add the listener.
7. Install the application.
8. Wait for some timeout so that the program does not end.
9. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
10. When the installation is done, remove the listener and quit.

Results

After you successfully run the code, the application is installed.

Example

The following example shows how to install an application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import com.ibm.websphere.management.application.*;
import com.ibm.websphere.management.application.client.*;
import com.ibm.websphere.management.*;

import javax.management.*;

public class Install {

    public static void main (String [] args) {

        try {
            String earFile = "C:/test/test.ear";
            String appName = "MyApp";

            // Preparation phase: Begin
            // Through the preparation phase you populate the enterprise archive (EAR) file with
            // WebSphere Application Server-specific binding information. For example, you can specify
            // Java Naming and Directory Interface (JNDI) names for enterprise beans, or virtual hosts
            // for Web modules, and so on.

            // First, create the controller and populate the EAR file with the appropriate options.
            Hashtable prefs = new Hashtable();
            prefs.put(AppConstants.APPDEPL_LOCALE, Locale.getDefault());

            // You can optionally run the default binding generator by using the following options.
            // Refer to Java documentation for the AppDeploymentController class to see all the
            // options that you can set.
            Properties defaultBnd = new Properties();
            prefs.put (AppConstants.APPDEPL_DFLTBN DG, defaultBnd);
            defaultBnd.put (AppConstants.APPDEPL_DFLTBN DG_VHOST, "default_host");

            // Create the controller.
            AppDeploymentController controller = AppDeploymentController
                .readArchive(earFile, prefs);
            AppDeploymentTask task = controller.getFirstTask();
```

```

    while (task != null)
    {
// Populate the task data.
    String[][] data = task.getTaskData();
// Manipulate task data which is a table of stringtask.
    task.setTaskData(data);
    task = controller.getNextTask();
    }
    controller.saveAndClose();

    Hashtable options = controller.getAppDeploymentSavedResults();
// The previous options table contains the module-to-server relationship if it was set by
// using tasks.
//Preparation phase: End

// Get a connection to WebSphere Application Server.
String host = "localhost";
String port = "8880";
String target = "WebSphere:cell=cellName,node=nodeName,server=server1";

Properties config = new Properties();
config.put (AdminClient.CONNECTOR_HOST, host);
config.put (AdminClient.CONNECTOR_PORT, port);
config.put (AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println ("Config: " + config);
    AdminClient _soapClient = AdminClientFactory.createAdminClient(config);

// Create the application management proxy, AppManagement.
AppManagement proxy = AppManagementProxy.getJMXProxyForClient (_soapClient);

// If code for the preparation phase has been run, then you already have the options table.
// If not, create a new table and add the module-to-server relationship to it by uncommenting
// the next statement.
//Hashtable options = new Hashtable();
    options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());

// Uncomment the following statements to add the module to the server relationship table if
// the preparation phase does not collect it.
//Hashtable module2server = new Hashtable();
//module2server.put ("*", target);
//options.put (AppConstants.APPDEPL_MODULE_TO_SERVER, module2server);

//Create the notification filter for listening to installation events.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
myFilter.enableType (AppConstants.NotificationType);

//Add the listener.
NotificationListener listener = new AListener(_soapClient,
myFilter, "Install: " + appName, AppNotification.INSTALL);

// Install the application.
proxy.installApplication (earFile, appName, options, null);
System.out.println ("After install App is called..");

// Wait for some timeout. The installation application programming interface (API) is
// asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
Thread.sleep(300000); // Wait so that the program does not end.

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient c1, NotificationFilterSupport f1,
Object h, String eType) throws Exception
    {
        _soapClient = c1;
        myFilter = f1;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName(
"WebSphere:type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
        AppNotification ev = (AppNotification) notf.getUserData();
        System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

        //When the installation is done, remove the listener and quit.

        if (ev.taskName.equals (eventTypeToCheck) &&
            (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
            ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
        {
            try
            {
                _soapClient.removeNotificationListener (on, this);
            }
            catch (Throwable th)
            {
                System.out.println ("Error removing listener: " + th);
            }
            System.exit (0);
        }
    }
}
}

```

What to do next

Once you install the application, you must explicitly start the application or you must stop and restart the server. For information on starting an application, see the *Starting an application through programming* topic in the *Using the administrative clients* PDF. For information on stopping or restarting the server, see the *Stopping an application server* topic or the *Starting an application server* topic, respectively, in the *Setting up the application serving environment* PDF.

Starting an application through programming

You can start an application through the administrative console, the wsadmin tool, or programming. Use this example to start an application through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can start an application on WebSphere Application Server, you must first install your application.

About this task

Perform the following tasks to start an application through programming.

1. Connect the administrative client to WebSphere Application Server.
2. Create the application management proxy.
3. Call the startApplication method on the proxy by passing the application name and optionally the list of targets on which to start the application.

Results

After you successfully run the code, the application is started.

Example

The following example shows how to start an application following the previously listed steps. Some statements are split on multiple lines for printing purposes.

```
//Do a get of the administrative client to connect to
//WebSphere Application Server.

AdminClient client = ...;
String appName = "myApp";
Hashtable prefs = new Hashtable();
// Use the AppManagement MBean to start and stop applications on all or some targets.

// The AppManagement MBean is on server1 in WebSphere Application Server.
// Query and get the AppManagement MBean.
ObjectName on = new ObjectName ("WebSphere:type=AppManagement,*");
Iterator iter = client.queryNames (on, null).iterator();
ObjectName appmgmtON = (ObjectName)iter.next();

//Start the application on all targets.
AppManagement proxy = AppManagementProxy.getJMXProxyForClient(client);
String started = proxy.startApplication(appName, prefs, null);
System.out.println("Application started on folloing servers: " + started);

//Start the application on some targets.
//String targets = "WebSphere:cell=cellname,node=nodename,
server=servername+WebSphere:cell=cellname,cluster=clusterName";
//String started1 = proxy.startApplication(appName, targets, prefs, null);
//System.out.println("Application started on following servers: " + started1)
```

Uninstalling an application through programming

You can uninstall an application through the administrative console, the wsadmin tool, or programming. Use this example to uninstall an application through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can uninstall an application on WebSphere Application Server, you must first install it.

About this task

Perform the following tasks to uninstall an application through programming.

1. Get a connection to WebSphere Application Server.
2. Get the application management proxy.
3. Create the notification filter for listening to uninstallation events.
4. Add the listener.
5. Uninstall the application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the uninstallation is done, remove the listener and quit.

Results

After you successfully run the code, the application is uninstalled.

Example

The following example shows how to uninstall an application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import com.ibm.websphere.management.application.*;
import com.ibm.websphere.management.application.client.*;
import com.ibm.websphere.management.*;

import javax.management.*;

public class Uninstall {

    public static void main (String [] args) {

        try {

// Get a connection to the server.
String host = "localhost";
String port = "8880";
String target = "WebSphere:cell=cellName,node=nodeName,server=server1";

Properties config = new Properties();
config.put (AdminClient.CONNECTOR_HOST, host);
config.put (AdminClient.CONNECTOR_PORT, port);
config.put (AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println ("Config: " + config);
    AdminClient _soapClient = AdminClientFactory.createAdminClient(config);

// Get the application management proxy.
AppManagement proxy = AppManagementProxy.getJMXProxyForClient (_soapClient);

String appName = "MyApp";
Hashtable options = new Hashtable();
options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());

//Create the notification filter.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
```

```

myFilter.enableType (AppConstants.NotificationType);

//Add the listener.
NotificationListener listener = new AListener(_soapClient,
myFilter, "Install: " + appName, AppNotification.UNINSTALL);

// Uninstall the application.
proxy.uninstallApplication (appName, options, null);
System.out.println ("After uninstall App is called..");

// Wait for some timeout. The installation application programming interface (API) is
// asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
Thread.sleep(300000); // Wait so that the program does not end.

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl,
Object h, String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName(
"WebSphere:type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
        AppNotification ev = (AppNotification) notf.getUserData();
        System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

        //When the unistallation is done, remove the listener and quit

        if (ev.taskName.equals (eventTypeToCheck) &&
            (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
            ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
        {
            try
            {
                _soapClient.removeNotificationListener (on, this);
            }
            catch (Throwable th)
            {

```

```

        System.out.println ("Error removing listener: " + th);
    }
    System.exit (0);
}
}
}

```

Manipulating additional attributes for a deployed application

You can manipulate attributes for a deployed application through the administrative console, the wsadmin tool, or by programming. Use this example to manipulate attributes that are not exposed during or after application installation through the AppDeploymentTask object.

Before you begin

This task assumes a basic familiarity with MBean programming and the ConfigService interfaces. Read about MBean programming and ConfigService interfaces in the application programming interfaces documentation.

About this task

Perform the following tasks for your deployed application to manipulate attributes that are not exposed through the AppDeploymentTask object. The attributes are saved in the deployment.xml file that is created in the configuration repository for each deployed application.

1. Create a session.
2. Connect to WebSphere Application Server.
3. Locate the ApplicationDeployment object.
4. Manipulate the attributes.
5. Save your changes.
6. Clean up the session.

Results

After you successfully run the code, the attributes are updated in the deployment.xml file for the deployed application.

Example

The following example shows how to manipulate the startingWeight, warClassLoaderPolicy, and classloader attributes based on the previous steps.

```

import java.util.Properties;

import javax.management.Attribute;
import javax.management.AttributeList;
import javax.management.ObjectName;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.configservice.ConfigService;
import com.ibm.websphere.management.configservice.ConfigServiceHelper;
import com.ibm.websphere.management.configservice.ConfigServiceProxy;
import com.ibm.websphere.management.exception.ConfigServiceException;
import com.ibm.websphere.management.exception.ConnectorException;

public class AppManagementSample1 {

    public static void main(String[] args) {

```

```

String hostName = "localhost";
String port = "8880";
String appName = "ivtApp";

ConfigService configService = null;

// create a session.
Session session = new Session();

// establish connection to the server.
try {
    Properties props = new Properties();
    props.setProperty(AdminClient.CONNECTOR_TYPE,
        AdminClient.CONNECTOR_TYPE_SOAP);
    props.setProperty(AdminClient.CONNECTOR_HOST, hostName);
    props.setProperty(AdminClient.CONNECTOR_PORT, port);
    AdminClient adminClient =
        AdminClientFactory.createAdminClient(props);

    // create a config service proxy object.
    configService = new ConfigServiceProxy(adminClient);

    // Locate the application object.
    ObjectName rootID = configService.resolve(session,
        "Deployment="+appName)[0];
    System.out.println ("rootID is: " + rootID);

    // Locate the ApplicationDeployment object from the root.
    ObjectName appDeplPattern = ConfigServiceHelper.createObjectName
        (null, "ApplicationDeployment");
    /*
    ObjectName appDeplID = configService.queryConfigObjects(session,
        rootID, appDeplPattern, null)[0];
    */
    AttributeList list1 = configService.getAttributes(session,
        rootID, new String[]{"deployedObject"}, false);
    ObjectName appDeplID = (ObjectName)
        ConfigServiceHelper.getAttributeValue(list1, "deployedObject");
    System.out.println ("appDeplID: " + appDeplID);

    // Locate the class loader.

    // Change the starting weight through the startingWeight attribute. The starting weight
    // affects the order in which applications start.
    AttributeList attrList = new AttributeList();
    Integer newWeight = new Integer (10);
    attrList.add(new Attribute("startingWeight", newWeight));

    // Change the WAR class loader policy through the warClassLoaderPolicy attribute by
    // specifying SINGLE or MULTIPLE.
    // SINGLE=one classloader for all WAR modules
    attrList.add(new Attribute("warClassLoaderPolicy", "SINGLE"));

    // Set the class loader mode to PARENT_FIRST or PARENT_LAST.
    AttributeList clList = (AttributeList) configService.getAttribute
        (session, appDeplID, "classloader");
    ConfigServiceHelper.setAttributeValue (clList, "mode",
        "PARENT_LAST");
    attrList.add (new Attribute ("classloader", clList));

    // Set the new values.
    configService.setAttributes(session, appDeplID, attrList);

    // Save your changes.
    configService.save(session, false);

} catch (Exception ex) {

```



```

    ex.printStackTrace();
} finally {
    // Clean up the session.
    try {
        configService.discard(session);
    }
    catch (ConfigServiceException csEx)
    {
        csEx.printStackTrace();
    }
    catch (ConnectorException cnEx)
    {
        cnEx.printStackTrace();
    }
}
}
}
}

```

Sharing sessions for application management

With the configuration service interface, `ConfigService`, you can create a session that is a temporary staging area, where you can save all the configuration modifications. Saving the session saves all the updates from the session into the WebSphere Application Server configuration repository. The application management logic supports session sharing with the configuration service. You can perform all the application management functions in the same session as the one that the configuration service creates. Saving such a session saves all the updates, including the ones that are application-specific.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming, see *MBean Java application programming interface (API) documentation*.

About this task

Perform the following tasks for your deployed application to share and save application-specific updates through the `configService` configuration service.

1. Create a configuration service proxy object.
2. Create a session.
3. Pass the session information to the `AppManagement` MBean.

Every method on the `AppManagement` interface takes session ID (workspace ID) as the last parameter. If the session information is passed in this parameter, the application management function uses the session. If you set the parameter to a null value:

- No session sharing occurs
- The configuration changes are always saved in the configuration repository if the operation succeeds.

4. Save the session after all the necessary changes are made.

The following example outlines the general steps for session sharing through the `configService` configuration service. For a detailed example, see “Manipulating additional attributes for a deployed application” on page 331.

```

public void installApplication (String localEarPath,
                               String appName, Hashtable properties, String workspaceID)
    throws AdminException;

AdminClient adminClient = ....;

// Create a configuration service proxy object.
ConfigService configService = new ConfigServiceProxy(adminClient);

```

```

// Create a session.
Session session = new Session();

// Pass the session information to AppManagement MBean.
appMgmt = ...
appMgmt.installApplication
    (earPath, appName, properties, session.toString());
//Save the session after all necessary changes are made.
configService.save(session, false);

```

Results

After you successfully complete the steps, you have saved application-specific updates for a deployed application to a session, and then to the configuration repository.

Updating an application through programming

You can update an existing application through the administrative console, the wsadmin tool, or programming. Use this example to completely update an application through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can update an application on WebSphere Application Server, you must first install your application.

About this task

Perform the following tasks to completely update an application through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Prepare the enterprise archive (EAR) file by populating it with binding information.
6. Update the application.
7. Wait for some timeout so that the program does not end.
8. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
9. When the update is done, remove the listener and quit.

Results

After you successfully run the code, the application is updated.

Example

The following example shows how to update an application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```

import java.lang.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import com.ibm.websphere.management.application.*;
import com.ibm.websphere.management.application.client.*;
import com.ibm.websphere.management.*;

```

```

import javax.management.*;

public class aa {

    public static void main (String [] args) {

        try {

            // Connect to WebSphere Application Server.
            String host = "localhost";
            String port = "8880";
            String target = "WebSphere:cell=cellName,node=nodeName,server=server1";

            Properties config = new Properties();
            config.put (AdminClient.CONNECTOR_HOST, host);
            config.put (AdminClient.CONNECTOR_PORT, port);
            config.put (AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println ("Config: " + config);
            AdminClient _soapClient = AdminClientFactory.createAdminClient(config);

            // Create the application management proxy, AppManagement.
            AppManagement proxy = AppManagementProxy.getJMXProxyForClient (_soapClient);

            String appName = "MyApp";
            String fileContents = "/test/test.ear";

            // Create the notification filter.
            NotificationFilterSupport myFilter = new NotificationFilterSupport();
            myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
            //Add the listener.
            NotificationListener listener = new AListener(_soapClient, myFilter,
            "Install: " + appName, AppNotification.INSTALL);

            // Refer to the installation example to see how you can prepare the enterprise archive (EAR)
            // file by populating it with binding information.
            // If code for the preparation phase has started, then you already have the options table.
            // If not, create a new table and add the module-to-server relationship to it by uncommenting
            // the next statement.
            //Hashtable options = new Hashtable();
            options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
            options.put ((AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_APP);

            // Uncomment the following statements to add the module to the server relationship table if
            // the preparation phase does not collect it
            //Hashtable module2server = new Hashtable();
            //module2server.put ("*", target);
            //options.put (AppConstants.APPDEPL_MODULE_TO_SERVER, module2server);
            // Update the application.
            proxy.updateApplication (    appName,
                null,
                fileContents,
                AppConstants.APPUPDATE_UPDATE,
                options,
                null);

            // Wait for some timeout. The installation application programming interface (API) is
            // asynchronous and so returns immediately.
            // If the program does not wait here, the program ends.
            Thread.sleep(300000); // Wait so that the program does not end.

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl,
Object h, String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName(
"WebSphere:type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
        AppNotification ev = (AppNotification) notf.getUserData();
        System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

        //When the installation is done, remove the listener and quit

        if (ev.taskName.equals (eventTypeToCheck) &&
            (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
            ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
        {
            try
            {
                _soapClient.removeNotificationListener (on, this);
            }
            catch (Throwable th)
            {
                System.out.println ("Error removing listener: " + th);
            }
            System.exit (0);
        }
    }
}
}

```

Adding to, updating, or deleting part of an application through programming

You can add to, update, or delete part of an existing application through the administrative console, the wsadmin tool, or programming. This example changes part of an application through programming. You can use this example whether you add to, update, or delete part of an existing application. Multiple changes to an application can be packaged in a single .zip file.

Before you begin

To learn about the structure of the .zip file, see the Updating applications topic in the *Developing and deploying applications* PDF.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can add to, update, or delete part of an application on WebSphere Application Server, you must first install your application.

About this task

Perform the following tasks to add to, update, or delete part of an application through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter.
4. Add the listener.
5. Partially change the existing application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the update is done, remove the listener and quit.

Results

After you successfully run the code, you have changed the application.

Example

The following example shows how to add to, update, or delete part of an application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
//Inputs:
//partialApp specifies the location of the partial application.
//appName specifies the name of the application.
String partialApp = "/apps/partial.zip";
String appName = "MyApp";

//Do a get of the administrative client to connect to
//WebSphere Application Server.

AdminClient client = ...;

//Create the application management proxy.
AppManagement proxy = AppManagementProxy.getJMXProxyForClient (client);

// Create the notification filter.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
//Add the listener.
NotificationListener listener = new AListener(_soapClient, myFilter,
"Install: " + appName, AppNotification.UPDATE);
//Partially change the existing application, MyApp.

Hashtable options = new Hashtable();
options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_PARTIALAPP);

proxy.updateApplication ( appName,
    null,
    partialApp,
    null,
    options,
    null);
```

```

// Wait for some timeout. The installation application programming interface (API) is
// asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
Thread.sleep(300000); // Wait so that the program does not end.
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl,
Object h, String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName(
"WebSphere:type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
        AppNotification ev = (AppNotification) notf.getUserData();
        System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

        //When the installation is done, remove the listener and quit

        if (ev.taskName.equals (eventTypeToCheck) &&
            (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
            ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
        {
            try
            {
                _soapClient.removeNotificationListener (on, this);
            }
            catch (Throwable th)
            {
                System.out.println ("Error removing listener: " + th);
            }
            System.exit (0);
        }
    }
}
}

```

Editing applications

You can edit deployed applications through the administrative console, the wsadmin tool, or by programming. Use this example to edit a deployed application through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming, see MBean Java application programming interface (API) documentation.

Before you can edit an application on WebSphere Application Server, you must first install the application.

About this task

Perform the following tasks to edit your deployed application.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Get information about an installed application.
4. Manipulate task data as necessary.
5. Save changes for the installed application.

Results

After you successfully run the code, the application is edited.

Example

The following example shows how to edit an application, based on the previous steps.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import com.ibm.websphere.management.application.*;
import com.ibm.websphere.management.application.client.*;
import com.ibm.websphere.management.*;

import javax.management.*;

public class aa {

    public static void main (String [] args) {

        try {

            // Connect to WebSphere Application Server.
            String host = "localhost";
            String port = "8880";
            String target = "WebSphere:cell=cellName,node=nodeName,server=server1";

            Properties config = new Properties();
            config.put (AdminClient.CONNECTOR_HOST, host);
            config.put (AdminClient.CONNECTOR_PORT, port);
            config.put (AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println ("Config: " + config);
            AdminClient _soapClient = AdminClientFactory.createAdminClient(config);

            // Create the application management proxy, AppManagement.
            AppManagement proxy = AppManagementProxy.getJMXProxyForClient (_soapClient);

            String appName = "MyApp";
            // Get information for an application with name appName:
            // Pass Locale information as the preference.
            Hashtable prefs = new Hashtable();
            prefs.put(AppConstants.APPDEPL_LOCALE, Locale.getDefault());
            Vector allTasks = appMgmt.getApplicationInfo (appName, prefs, null);
```

```

// Manipulate task data as necessary.
if (task.getName().equals ("MapRolesToUsers") && !task. isTaskDisabled())
{
    // find out column index for role and user column
    // refer to the previous table to find the column names
    int roleColumn = -1;
    int userColumn = -1;
    String[] colNames = task.getColumnNames();
    for (int i=0; i < colNames.length; i++)
    {
        if (colNames[i].equals ("role"))
            roleColumn = i;
        else if (colNames[i].equals ("role.user"))
            userColumn = i;
    }

    // iterate over task data starting at row 1 as row0 is
    // column names
    String[][] data = task.getTaskData();
    for (int i=1; i < data.length; i++)
    {
        if (data[i][roleColumn].equals ("Role1"))
        {
            data[i][userColumn]="User1|User2";
            break;
        }
    }

    // now that the task data is changed, save it back
    task.setTaskData (data);
}

// Save changes back into the installed application:
// Set information for an application with name appName.
// Pass Locale information as the preference.
prefs = new Hashtable();
prefs.put(AppConstants.APPDEPL_LOCALE, Locale.getDefault());
appMgmt.setApplicationInfo (appName, prefs, null, allTasks);

    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
}
}

```

Preparing a module and adding it to an existing application through programming

You can add a module to an existing application through the administrative console, the wsadmin tool, or programming. Use this example to add a module through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can add a module to an application on WebSphere Application Server, you must install the application.

About this task

Perform the following tasks to add a module to an application through programming.

1. Create an application deployment controller instance to populate the module file with binding information.
2. Save the binding information in the module.
3. Get the installation options.
4. If the preparation phase (population of the EAR file) is not performed, then do the following actions:
 - a. Create an options table to be passed to the updateApplication MBean API.
 - b. Create a table for module to server relations and add the table to the options table.
5. Connect to WebSphere Application Server.
6. Create the application management proxy.
7. Create the notification filter.
8. Add the listener.
9. Add the module to the application.
10. Specify the target for the new module.
11. Wait for some timeout so that the program does not end.
12. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
13. When the module addition is done, remove the listener and quit.

Results

After you successfully run the code, the module is added to the application.

Example

The following example shows how to add a module to an application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
//Inputs:
//moduleName specifies the name of the module that you add to the application.
//moduleURI specifies a URI that gives the target location of the module
// archive contents on a file system. The URI provides the location of the new
// module after installation. The URI is relative to the application URL.
//uniquemoduleURI specifies the URI that gives the target location of the
// deployment descriptor file. The URI is relative to the application URL.
//target specifies the cell, node, and server on which the module is installed.
String moduleName = "/apps/foo.jar";
String moduleURI = "Increment.jar";
String uniquemoduleURI = "Increment.jar+META-INF/ejb-jar.xml";
String target = "WebSphere:cell=cellname,node=nodename,server=servername";

//Create an application deployment controller instance, AppDeploymentController,
//to populate the Java Archive (JAR) file with binding information.
//The binding information is WebSphere Application Server-specific deployment information.

Hashtable preferences = new Hashtable();
preferences.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
preferences.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_MODULEFILE);
AppDeploymentController controller = AppManagementFactory.readArchiveForUpdate(
    moduleName,
    moduleURI,
    AppConstants.APPUPDATE_ADD,
    preferences,
    null);
```

If the module that you add to the application lacks any bindings, add the bindings so that the module addition works. Collect and add the bindings by using the public APIs provided with WebSphere Application Server. Refer to Java documentation for the `com.ibm.websphere.management.application.client.AppDeploymentController` instance to learn more about how to collect and populate tasks with WebSphere Application Server-specific binding information. The `AppDeploymentController` instance contains meta-data defined in XML-based deployment descriptors as well as annotations defined in Java classes within the input module.

```
//After you collect all the binding information, save it in the module.
controller.saveAndClose();

//Get the installation options.
Hashtable options = controller.getAppDeploymentSavedResults();

//Connect the administrative client, AdminClient, to WebSphere Application Server.
AdminClient client = ...;

//Create the application management proxy.
AppManagement proxy = AppManagementProxy.getJMXProxyForClient (client);

//Update the existing application, MyApp, by adding the module.
String appName = "MyApp";

options.put (AppConstants.APPUPDATE_CONTENTTYPE,
    AppConstants. APPUPDATE_CONTENT_MODULEFILE);

//Create the notification filter.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
//Add the listener.
NotificationListener listener = new AListener(_soapClient, myFilter,
"Install: " + appName, AppNotification.UPDATE);

//Specify the target for the new module.
Hashtable mod2svr = new Hashtable();
options.put (AppConstants.APPDEPL_MODULE_TO_SERVER, mod2svr);
mod2svr.put (uniquemoduleURI, target);
proxy.updateApplication ( appName,
    moduleURI,
    moduleName,
    AppConstants.APPUPDATE_ADD,
    options,
    null);

// Wait for some timeout. The installation application programming interface (API) is
// asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
Thread.sleep(300000); // Wait so that the program does not end.
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;
}
```

```

    public AListener(AdminClient c1, NotificationFilterSupport fl,
Object h, String eType) throws Exception
    {
        _soapClient = c1;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName(
"WebSphere:type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
        AppNotification ev = (AppNotification) notf.getUserData();
        System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

        //When the installation is done, remove the listener and quit

        if (ev.taskName.equals (eventTypeToCheck) &&
            (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
             ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
        {
            try
            {
                _soapClient.removeNotificationListener (on, this);
            }
            catch (Throwable th)
            {
                System.out.println ("Error removing listener: " + th);
            }
            System.exit (0);
        }
    }
}

```

Preparing and updating a module through programming

You can update a module for an existing application through the administrative console, the wsadmin tool, or programming. When you update a module, you replace the existing module with a new version. Use this example to update a module through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can update a module on WebSphere Application Server, you must first install the application.

About this task

Perform the following tasks to update a module through programming.

1. Create an application deployment controller instance to populate the Java archive file with binding information.
2. Save the binding information in the module.
3. Get the installation options.

4. If the preparation phase (population of the EAR file) is not performed, the do the following actions:
 - a. Create an options table to be passed to the updateApplication MBean API.
 - b. Create a table for module to server relations and add the table to the options table.
5. Connect to WebSphere Application Server.
6. Create the application management proxy.
7. Create the notification filter.
8. Add the listener.
9. Replace the module in the application.
10. Specify the target for the new module.
11. Wait for some timeout so that the program does not end.
12. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
13. When the module addition is done, remove the listener and quit.

Results

After you successfully run the code, the existing module is replaced with the new one.

Example

The following example shows how to add a module to an application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
//Inputs:
//moduleName specifies the name of the module that you add to the application.
//moduleURI specifies a URI that gives the target location of the module
// archive contents on a file system. The URI provides the location of the new
// module after installation. The URI is relative to the application URL.
//uniquemoduleURI specifies the URI that gives the target location of the
// deployment descriptor file. The URI is relative to the application URL.
//target specifies the cell, node, and server on which the module is installed.
//appName specifies the name of the application to update.
String moduleName = "/apps/foo.jar";
String moduleURI = "Increment.jar";
String uniquemoduleURI = "Increment.jar+META-INF/ejb-jar.xml";
String target = "WebSphere:cell=cellname,node=nodename,server=servername";
String appName = "MyApp";

//Get the administrative client to connect to
//WebSphere Application Server.
AdminClient client = ...;
AppManagement proxy = AppManagementProxy.getJMXProxyForClient (client);

Vector tasks = proxy.getApplicationInfo (appName, new Hashtable(), null);

//Create an application deployment controller instance, AppDeploymentController,
//to populate the Java archive (JAR) file with binding information.
//The binding information is WebSphere Application Server-specific deployment information.

Hashtable preferences = new Hashtable();
preferences.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
preferences.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_MODULEFILE);
AppDeploymentController controller = AppManagementFactory.readArchiveForUpdate(
    moduleName,
    moduleURI,
    AppConstants.APPUPDATE_UPDATE,
    preferences,
    tasks);
```

If the module that you update for the application lacks any bindings, add the bindings so that the module update works. Collect and add the bindings by using the public APIs that are provided with WebSphere Application Server. Refer to Java documentation for the `AppDeploymentController` instance to learn more about how to collect and populate tasks with WebSphere Application Server-specific binding information. The `AppDeploymentController` instance contains meta-data defined in XML-based deployment descriptors as well as annotations defined in Java classes within the input module.

```
//After you collect all the binding information, save it in the module.
controller.saveAndClose();

//Create the notification filter.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
//Add the listener.
NotificationListener listener = new AListener(_soapClient, myFilter,
"Install: " + appName, AppNotification.UPDATE);

//Get the installation options.
Hashtable options = controller.getAppDeploymentSavedResults();

//Update the existing application by adding the module.

options.put (AppConstants.APPUPDATE_CONTENTTYPE,
AppConstants.APPUPDATE_CONTENT_MODULEFILE);

//Specify the target for the new module
Hashtable mod2svr = new Hashtable();
options.put (AppConstants.APPDEPL_MODULE_TO_SERVER, mod2svr);
mod2svr.put (uniquemoduleURI, target);

proxy.updateApplication ( appName,
moduleURI,
moduleName,
AppConstants.APPUPDATE_UPDATE,
options,
null);
// Wait. The installation application programming interface (API) is
// asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
Thread.sleep(300000); // Wait so that the program does not end.
}
catch (Exception e) {
e.printStackTrace();
}
}

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
AdminClient _soapClient;
NotificationFilterSupport myFilter;
Object handback;
ObjectName on;
String eventTypeToCheck;

public AListener(AdminClient c1, NotificationFilterSupport fl,
Object h, String eType) throws Exception
{
_soapClient = c1;
myFilter = fl;
handback = h;
eventTypeToCheck = eType;

Iterator iter = _soapClient.queryNames (new ObjectName(
```

```

"WebSphere:type=AppManagement,*"), null).iterator();
    on = (ObjectName)iter.next();
    System.out.println ("ObjectName: " + on);
    _soapClient.addNotificationListener (on, this, myFilter, handback);
}

public void handleNotification (Notification notf, Object handback)
{
    AppNotification ev = (AppNotification) notf.getUserData();
    System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

    //When the installation is done, remove the listener and quit

    if (ev.taskName.equals (eventTypeToCheck) &&
        (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
         ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
    {
        try
        {
            _soapClient.removeNotificationListener (on, this);
        }
        catch (Throwable th)
        {
            System.out.println ("Error removing listener: " + th);
        }
        System.exit (0);
    }
}
}
}

```

Deleting a module through programming

You can delete a module from an existing application through the administrative console, the wsadmin tool, or programming. Use this example to delete a module through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can delete a module from an application on WebSphere Application Server, you must first install the application.

About this task

Perform the following tasks to delete a module through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Delete the module.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the module is deleted, remove the listener and quit.

Results

After you successfully run the code, the existing module is deleted from the application.

Example

The following example shows how to delete a module from an application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
//moduleURI specifies a URI that gives the target location of the module.
//appName specifies the name of the application to update.
String moduleURI = "Increment.jar";
String appName = "MyApp";

//Get the administrative client to connect to
//WebSphere Application Server.
AdminClient client = ...;

//Create the application management proxy.

AppManagement proxy = AppManagementProxy.getJMXProxyForClient (client);

//Create the notification filter.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
//Add the listener.
NotificationListener listener = new AListener(_soapClient, myFilter,
"Install: " + appName, AppNotification.UPDATE);

//Update the existing application, MyApp, by deleting the module.
Hashtable options = new Hashtable();
options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_MODULEFILE);

proxy.updateApplication ( appName,
    moduleURI,
    null,
    AppConstants.APPUPDATE_DELETE,
    options,
    null);

// Wait; the installation application programming interface (API) is
// asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
Thread.sleep(300000); // Wait so that the program does not end.
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl,
Object h, String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName(
```

```

"WebSphere:type=AppManagement,*"), null).iterator();
    on = (ObjectName)iter.next();
    System.out.println ("ObjectName: " + on);
    _soapClient.addNotificationListener (on, this, myFilter, handback);
}

public void handleNotification (Notification notf, Object handback)
{
    AppNotification ev = (AppNotification) notf.getUserData();
    System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

    //When the installation is done, remove the listener and quit

    if (ev.taskName.equals (eventTypeToCheck) &&
        (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
         ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
    {
        try
        {
            _soapClient.removeNotificationListener (on, this);
        }
        catch (Throwable th)
        {
            System.out.println ("Error removing listener: " + th);
        }
        System.exit (0);
    }
}
}
}

```

Adding a file through programming

You can add a file to an existing application through the administrative console, the wsadmin tool, or programming. This example describes how to add a file through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can add a file to an application on WebSphere Application Server, you must first install the application.

About this task

Perform the following tasks to add a file to an application through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Add the file to the application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the file is added to the application, remove the listener and quit.

Results

After you successfully run the code, the file is added to the application.

Example

The following example shows how to add a file to an application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import com.ibm.websphere.management.application.*;
import com.ibm.websphere.management.application.client.*;
import com.ibm.websphere.management.*;

import javax.management.*;

public class FileAdd {

    public static void main (String [] args) {

        try {

// Get a connection to WebSphere Application Server.
String host = "localhost";
String port = "8880";
String target = "WebSphere:cell=cellName,node=nodeName,server=server1";

Properties config = new Properties();
config.put (AdminClient.CONNECTOR_HOST, host);
config.put (AdminClient.CONNECTOR_PORT, port);
config.put (AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println ("Config: " + config);
    AdminClient _soapClient = AdminClientFactory.createAdminClient(config);

// Create the application management proxy, AppManagement.
AppManagement proxy = AppManagementProxy.getJMXProxyForClient (_soapClient);

String appName = "MyApp";
String fileURI = "test.war/com/acme/abc.jsp";
String fileContents = "/temp/abc.jsp";

//Create the notification filter.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);

//Add the listener.
NotificationListener listener = new AListener(_soapClient, myFilter,
"Install: " + appName, AppNotification.UPDATE);

Hashtable options = new Hashtable();
options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_FILE);

// Update the application
proxy.updateApplication (    appName,
                            fileURI,
                            fileContents,
                            AppConstants.APPUPDATE_ADD,
                            options,
                            null);

// Wait; the installation Application Programming Interface (API) is
// asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
Thread.sleep(90000); // Wait so that the program does not end.

        }
        catch (Exception e) {
```

```

        e.printStackTrace();
    }
}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl,
Object h, String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName(
"WebSphere:type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
        AppNotification ev = (AppNotification) notf.getUserData();
        System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

        //When the installation is done, remove the listener and quit

        if (ev.taskName.equals (eventTypeToCheck) &&
            (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
            ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
        {
            try
            {
                _soapClient.removeNotificationListener (on, this);
            }
            catch (Throwable th)
            {
                System.out.println ("Error removing listener: " + th);
            }
            System.exit (0);
        }
    }
}
}

```

Updating a file through programming

You can update a file for an existing application through the administrative console, the wsadmin tool, or programming. This example describes how to update a file through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can update a file for an application on WebSphere Application Server, you must first install the application.

About this task

Perform the following tasks to update a file through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Update the file in the application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the installation is done, remove the listener and quit.

Results

After you successfully run the code, the file is updated for the application.

Example

The following example shows how to add a file to an application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
//Inputs:
//fileContents specifies the name of the file that you add to the application.
//appName specifies the name of the application.
//fileURI specifies a URI that gives the target location of the file. The URI
// provides the location of the new module after installation. The URI is
// relative to the application URL.
String fileContents = "/apps/test.jsp";
String appName = "MyApp";
String fileURI = "SomeWebMod.war/com/foo/abc.jsp";

//Get the administrative client to connect to
//WebSphere Application Server.
AdminClient client = ...;

//Create the application management proxy.
AppManagement proxy = AppManagementProxy.getJMXProxyForClient (client);

//Create the notification filter.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
//Add the listener.
NotificationListener listener = new AListener(_soapClient, myFilter,
"Install: " + appName, AppNotification.UPDATE);

Hashtable options = new Hashtable();
options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_FILE);

proxy.updateApplication ( appName,
fileURI,
```

```

fileContents,
AppConstants.APPUPDATE_UPDATE,
options,
null);

// Wait; the installation application programming interface (API) is
// asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
Thread.sleep(300000); // Wait so that the program does not end.
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl,
Object h, String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName(
"WebSphere:type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
        AppNotification ev = (AppNotification) notf.getUserData();
        System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

        //When the installation is done, remove the listener and quit.

        if (ev.taskName.equals (eventTypeToCheck) &&
            (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
            ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
        {
            try
            {
                _soapClient.removeNotificationListener (on, this);
            }
            catch (Throwable th)
            {
                System.out.println ("Error removing listener: " + th);
            }
            System.exit (0);
        }
    }
}
}

```

Deleting a file through programming

You can delete a file from an existing application through the administrative console, the wsadmin tool, or programming. Use this example to delete a file through programming.

Before you begin

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can delete a file from an application on WebSphere Application Server, you must first install the application.

About this task

Perform the following tasks to delete a file through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Delete the file from the application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the file is deleted from the application, remove the listener and quit.

Results

After you successfully run the code, the file is deleted from the application.

Example

The following example shows how to delete a file based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
//Inputs:
//fileURI specifies a URI that gives the target location of the file. The URI
// provides the location of the new module after installation. The URI is
// relative to the application URL.
//appName specifies the name of the application.

String fileURI = "Increment.jar/com/acme/Foo.class";
String appName = "MyApp";

//Get the administrative client to connect to
//WebSphere Application Server.
AdminClient client = ...;

//Create the application management proxy.
AppManagement proxy = AppManagementProxy.getJMXProxyForClient (client);

//Create the notification filter.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
//Add the listener.
NotificationListener listener = new AListener(_soapClient, myFilter,
"Install: " + appName, AppNotification.UPDATE);

//Update the existing application, MyApp, by deleting the file.
Hashtable options = new Hashtable();
```

```

options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_FILE);

proxy.updateApplication ( appName,
    fileURI,
    null,
    AppConstants.APPUPDATE_DELETE,
    options,
    null);

// Wait for some timeout. The installation Application Programming Interface (API) is
// asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
Thread.sleep(300000); // Wait so that the program does not end.
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl,
Object h, String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName(
"WebSphere:type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
        AppNotification ev = (AppNotification) notf.getUserData();
        System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);

        //Once the installation is done, remove the listener and quit

        if (ev.taskName.equals (eventTypeToCheck) &&
            (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
            ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
        {
            try
            {
                _soapClient.removeNotificationListener (on, this);
            }
            catch (Throwable th)
            {
                System.out.println ("Error removing listener: " + th);
            }
        }
    }
}

```

```

        System.exit (0);
    }
}

```

Extending application management operations through programming

You can use the common deployment framework to add additional logic to application management operations. The additional logic can do such tasks as code generation, configuration operations, additional validation, and so on. This topic demonstrates, through programming, how to plug into the *common deployment framework* to extend application management operations.

Before you begin

This task assumes a basic familiarity with Java application programming interfaces (APIs). Read about the Java APIs in the application programming interfaces documentation.

Before you can extend application management operations, you must first install WebSphere Application Server.

About this task

Use this example to extend application management through programming. The tasks that the extensions provide are available through all the administrative clients, such as the wsadmin tool, the administrative console, or through programmatic APIs that the AppManagement MBean provides.

1. Define your extension as an Eclipse plug-in and add a `plugin.xml` file to register your extension provider with the deployment framework.
 - a. In the `plugin.xml` file, provide an extension provider implementation class for the `common-deployment-framework-extensionprovider` extension point.
 - b. Put the plug-in Java archive (JAR) file in the `plugins` directory of your WebSphere Application Server installation.

```

<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="com.ibm.myproduct.MyExtensionProvider"
  name="My Extension"
  version="1.0.0">

  <extension point="common-deployment-framework-extensionprovider">
    <action class="com.acme.MyExtendProviderImpl"/>
  </extension>
</plugin>

```

2. Provide an extension provider.

An extension provider class provides steps for a given operation on an application Enterprise archive (EAR) file. Before an operation runs, the deployment framework queries all the registered extension providers for additional steps. A single list of steps is passed to each provider. Each provider can add steps to the list. The default provider that the deployment framework provides is called first to populate the list with default steps. Other extension providers are called next.

Various operations that you can extend through the common deployment framework are defined as constants in the `DeploymentConstants` class. These operations are described in the following table. Some operations are split on multiple lines for printing purposes.

Table 6. Extensible `DeploymentConstants` operations

Operation	Description
<code>DeploymentConstants.CDF_OP_INSTALLJ2EE</code>	Installs a Java Platform, Enterprise Edition (Java EE) EAR file

Table 6. Extensible DeploymentConstants operations (continued)

Operation	Description
DeploymentConstants.CDF_OP_EDITJ2EE	Edits a deployment application configuration
DeploymentConstants.CDF_OP_UPDATEJ2EE	Applies a fine-grained update to an application such as addition, removal, or update of a file or a module; or partial update of an application
DeploymentConstants.CDF_OP_UNINSTALLJ2EE	Uninstalls a Java EE application
DeploymentConstants.CDF_OP_CREATE_EAR_WRAPPERJ2EE	Wraps the contents input to the application installation into an EAR file

The AppManagement MBean, which is responsible for deploying and managing Java EE applications on WebSphere Application Server, runs all the operations except the CDF_OP_CREATE_EAR_WRAPPERJ2EE operation. Deploy the extensions that extend these operations in the plugins directory of the stand-alone Application Server .

Either the wsadmin utility or the administrative console runs the CDF_OP_CREATE_EAR_WRAPPERJ2EE operation when the input contents that are supplied to the CDF_OP_INSTALLJ2EE operation are not packaged as an EAR file. Deploy an extension that extends the CDF_OP_CREATE_EAR_WRAPPERJ2EE operation in the plugins directory of the wsadmin installation.

The following example provides an extension provider that does the following tasks:

- a. Adds two additional steps for the application installation operation
- b. Adds one step for wrapping input contents into an EAR file

package com.acme;

```
import com.ibm.websphere.management.deployment.registry.ExtensionProvider;
import com.ibm.websphere.management.deployment.core.DeploymentConstants;
```

```
public class MyExtensionProviderImpl extends ExtensionProvider {
    public void addSteps (String type, String op, String phase,
        List steps)
    {
        if (op.equals (DeploymentConstants.CDF_OP_INSTALLJ2EE))
        {
            // Add a code generation step.
            steps.add (0, new com.acme.CodeGenStep());
            // Add a configuration step.
            steps.add (new com.acme.ConfigStep());
        }
        else if (op.equals (DeploymentConstants.CDF_OP_CREATE_EAR_WRAPPERJ2EE))
        {
            // Add an ear-wrapper step.
            steps.add (new com.acme.EarWrapperStep());
        }
    }
}
```

3. Provide the deployment step implementation.

An extension provider adds a deployment step. The step contains logic that performs additional processing in an application management operation. The logic provides the step access to the deployment context and the deployable object. The deployment context provides information, such as the name of the operation, the configuration session ID, a temporary location for creating temporary files, operation parameters, and so on. The deployable object wraps the deployment content input to the operation. For example, the deployable object wraps the Java EE EAR file for the installation operation or a file, a module, or a partial application for the update operation.

- The following example illustrates how an extension during installation entirely changes an EAR file that is input to the installation operation. The example provides a deployment step during the installation operation that does the following tasks:

- a. Runs code generation to generate a new EAR file.
- b. Calls the `setContentPath` method in the `DeployableObject` class to set the new EAR file path. The default installation logic, such as steps that the default installation logic adds, uses this new EAR file for installation in the configuration repository.

```
package com.acme;

import com.ibm.websphere.management.deployment.core.DeploymentStep;
import com.ibm.websphere.management.deployment.core.DeployableObject;

public class CodeGenStep extends DeploymentStep
{
    public void execute (DeployableObject dObject)
    {
        EARFile earFile = (EARFile)dObject.getHandle();
        String newEARPath = null;
        // Use step specific logic to create another EAR file after code generation.
        ...
        newEARPath = _context.getTempDir() + "new.ear";

        dObject.setContentPath (newEARPath);
    }
}
```

- The following example provides a deployment step that:
 - a. Reads the contents of the input EAR file.
 - b. Manipulates the configuration session accessed through the context instance, `_context`.

```
package com.acme;

public class ConfigStep extends DeploymentStep
{
    public void execute (DeployableObject dObject)
    {
        EARFile earFile = (EARFile) dObject.getHandle();

        // Use the following example code to perform the configuration.
        String sessionID = _context.getSessionID();
        com.ibm.websphere.management.Session session = new
            com.ibm.websphere.management.Session (sessionID, true);
        // Use the configuration service to perform the configuration steps.
        ...

        // Read the application configuration.
        Application appDD = earFile.getDeploymentDescriptor();
        ...

        String newEARPath = null;
    }
}
```

- The following example provides a deployment step to wrap arbitrary content around an EAR file. Application management logic accepts only the EAR file for deployment. An extension is required if you want to input anything other than an EAR file to the deployment process.

```
package com.acme;

import com.ibm.websphere.management.deployment.core.DeploymentStep;
import com.ibm.websphere.management.deployment.core.DeployableObject;

public class EarWrapperStep extends DeploymentStep
{
    public void execute (DeployableObject dObject)
    {
        Archive archive = (Archive) dObject.getHandle();
        String newEARPath = null;
        // provide your logic to wrap the jar with the ear
        ...
    }
}
```

```
newEARPath = //;  
// Set the new ear path back into DeploymentContext  
this.getContext().getContextData()  
    .put(DeploymentContext.RETURN_Object_key, newEARPath);  
  
    }  
}
```

Results

Through programming, you have plugged into the common deployment framework to extend application management operations.

What to do next

You can extend other application management operations, or do any other administrative operations you choose.

Chapter 9. Deploying and administering business-level applications

Deploying a business-level application consists of creating the business-level application on a Version 7.0 or later server.

Before you begin

Note: A business-level application is an administration model that provides the entire definition of an application as it makes sense to the business. It is a WebSphere configuration artifact, similar to a server, that is stored in the product configuration repository. A business-level application can contain artifacts such as Java Platform, Enterprise Edition (Java EE) applications or modules, shared libraries, data files, and other business-level applications. You might use a business-level application to group related artifacts or to add capability to an existing application. For example, suppose you want to add capability provided in a Java archive (JAR) to a Java EE application already deployed on a product server. You can add that capability by creating a new business-level application and adding the JAR file and the deployed Java EE application to the business-level application. In some cases, you do not even need to change the deployed Java EE application configuration to add the capability.

Before creating a business-level application, you must develop the artifacts to go in the application and configure the target server. Before choosing a deployment target for the application, ensure that the target version is 7.0 or later.

About this task

When creating a business-level application, you can configure the application enough to enable it to run on the server. Later, you can configure the application and its contents further, start or stop the application, and otherwise manage its activity.

The topics in this section describe how to deploy and administer a business-level application or its contents using the administrative console. You can also use programming or wsadmin scripting.

- Import assets to a repository.
- View, delete, update, or export assets.
- Create a business-level application.
- Start the application.
- Stop the application.
- Update the application and its configuration units.
- Delete the application.

What to do next

After making changes to administrative configurations of your applications in the administrative console, ensure that you save the changes.

Business-level applications

A business-level application is an administration model that provides the entire definition of an application as it makes sense to the business. A business-level application is a WebSphere configuration artifact, similar to a server or cluster, that is stored in the product configuration repository.

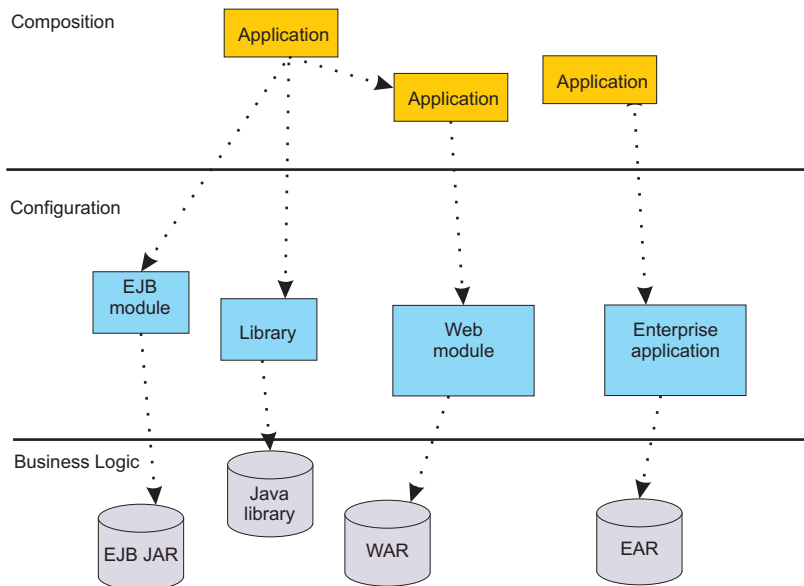
- Business-level application characteristics
- Comparisons to Java EE applications

Business-level application characteristics

A business-level application has the following characteristics:

- A business-level application is an administration model of the definition of an enterprise-level application that consists of WebSphere and non-WebSphere artifacts. The business-level application might not explicitly manage the lifecycle of every artifact. It is a model for defining an application.
- A business-level application does not represent or contain application binary files. It is a configuration that lists one or more composition units, which represent the application binary files. A business-level application uses the binary files to run the application business logic. Administration of binary files is separate from administration of the application definition.
- A business-level application supports recursive composition by reference that facilitates hierarchical assembly of business-level applications and individual deployed artifacts within or outside a WebSphere product. The composition at its lowest level consists of configured instances of application binary files that run in a specific runtime environment such as an application server. Installable packages or archives, such as Java archives (JAR) or enterprise archive (EAR) files, typically deliver the business logic that these configured instances represent to corresponding runtime platforms.

The following diagram shows the composition model for business-level applications:



A business-level application does not introduce new programming, runtime, or packaging models:

- You do not need to change your application business logic. The business-level application function does not introduce new application programming interfaces (APIs).
- You do not need to change your application runtime settings. The product supports all of the runtime characteristics, such as security, class loading and isolation, required by individual programming models to which business components are written.
- You do not need to change your application packaging. There is no specific unique packaging model that provides a business-level application definition.

Typically, you first create an empty business-level application and then add composition units to it. The business-level application name must be unique within a cell. The business level application itself has minimal configuration data associated with it, solely the list of composition units, but individual composition units might save application-specific configuration data.

A business-level application is defined in the product configuration repository under `app_server_root/cells/cell_name/blas/business_level_application_name/bver/BASE/bla.xml`.

Comparisons to Java EE applications

Business-level applications can consist of or aggregate Java Platform, Enterprise Edition (Java EE) applications and modules with non-Java EE artifacts. The contents of Java EE applications integrate with business-level application concepts for deployment and management of applications. Existing Java EE application management APIs continue to work after you add Java EE application or modules to a business-level application. The business-level application management API accepts Java EE contents and configurations and delegates to existing Java EE management APIs. Control operations such as starting and stopping a Java EE composition unit are delegated to ApplicationManager MBean on application servers that start and stop Java EE applications.

Table 7. Java EE concepts compared to business-level application concepts

Java EE concept	Business-level application concept	Description
EAR or stand-alone module for deployment	Asset	Java EE application contents are assets.
Java EE application created at the end of application install	Composition unit	A Java EE application is in an enterprise archive (EAR) file. The product saves the EAR file in the product repository as a composition unit.
Java EE modules within the EAR file	Deployable units in the asset	Each module in the EAR file is a deployable unit that you can install on independent deployment targets. The EAR file is still managed as a single asset in its entirety.
Java EE application installation using the administrative console, programming, or wsadmin commands	Multiple business-level application management commands During Java EE application deployment, you can specify the name of the business-level application to include the Java EE application. If the business-level application name is not set, the product creates a default business-level application with the same name as the Java EE application name. The product adds a composition unit with the same name as the Java EE application name under the business-level application. You can deploy multiple Java EE applications under a single business-level application.	You can make a Java EE application a business-level application and add it to another business-level application: <ol style="list-style-type: none"> 1. Install the Java EE application (EAR file) using the enterprise application installation console wizard, programming, or wsadmin. Keep the default selection to create a business-level application that has the same name as the Java EE application. 2. Create an empty business-level application. 3. Add the EAR file business-level application to the empty business-level application. The EAR file business-level application is a composition unit of the containing business-level application. Or, you can make a Java EE application an asset and add it to another business-level application: <ol style="list-style-type: none"> 1. Import an EAR file as an asset. It has an asset type aspect of Java EE ear. 2. Create an empty business-level application. 3. Add the Java EE application asset to the business-level application. The EAR file asset is a composition unit of the containing business-level application. 4. Collect targets for each deployable unit (Java EE module).
Uninstall Java EE application	Multiple business-level application management commands	You delete the Java EE application composition unit from the business-level application: <ol style="list-style-type: none"> 1. Remove the composition unit for the Java EE application from the business-level application. 2. If the EAR file is an asset, delete the asset.
Start the Java EE application.	Start the composition unit.	Starting a business-level application starts any Java EE application in it.

Table 7. Java EE concepts compared to business-level application concepts (continued)

Java EE concept	Business-level application concept	Description
Stop the Java EE application.	Stop the composition unit.	Stopping a business-level application stops any Java EE application in it.

Assets

An asset represents one or more application binary files that are stored in an asset repository. Typical assets include application business logic such as Java Platform, Enterprise Edition (Java EE) archives, library files, and other resource files.

An asset repository stores the binary files for the asset. The product configuration repository provides a default asset repository.

Assets in the configuration repository are managed by the product management domain. The configuration repository stores asset binary files in `app_server_root/config/cells/cell_name/assets/asset_name/aver/BASE/bin/`.

An asset name must be unique within a cell, the product administrative domain.

The product creates an `asset.xml` file when an asset is registered with the product configuration. The file contains information about the asset such as its name, destination location, and dependencies on other assets.

You must register files as assets before you can add them to one or more business-level applications. At the time of asset registration, you can import the physical application files into the product configuration repository or you can specify an external location where the asset resides.

Composition units

A composition unit represents a configured asset in a business-level application. A composition unit enables the asset contents to interact with other assets in the application. It also enables the product run time to load and run asset contents.

The product supports three types of composition units:

Asset composition units

Composition units created from assets by configuring each deployable unit of the asset to run on deployment targets.

Shared library composition units

Composition units created from JAR-based assets by ignoring all the deployable objects from the asset and treating the asset JAR file as a library of classes.

Business-level application composition units

Composition units created from business-level applications that are added to existing business-level applications.

A composition unit contains the following information:

- Configuration information that binds contents of an asset with a specific hosting run time and adds the configuration necessary for the run time to load and run the asset
- References to external services, components, or other resources that the asset uses
- Customized configurations for service definitions, references and other relevant configuration data

- A list of deployment targets or runtime environments along with the runtime environment-specific configuration where the composition unit runs.

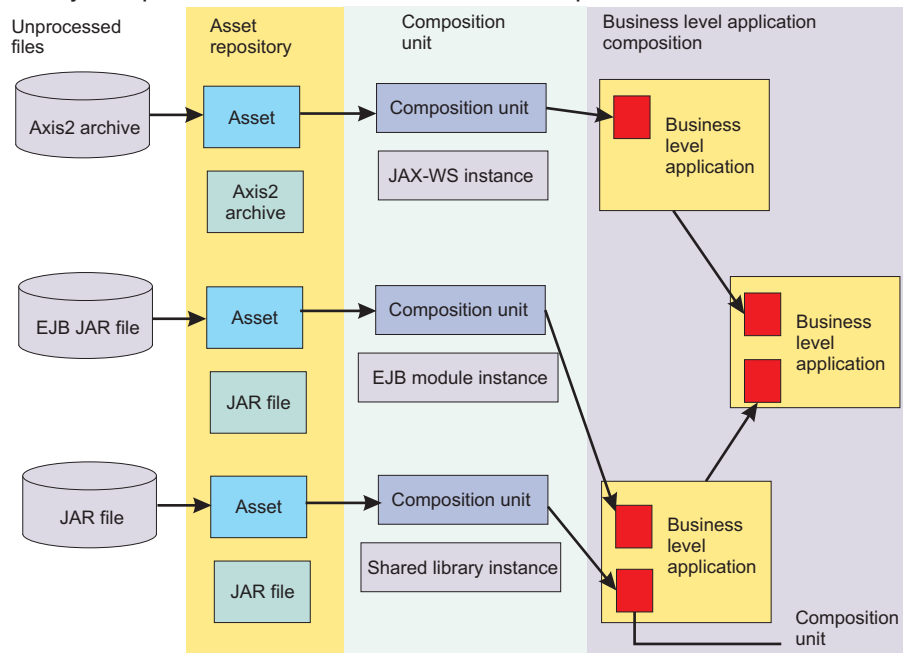
For example, a composition unit for an enterprise bean (EJB) Java archive (JAR) asset is an EJB module instance that contains necessary EJB binding information, such as EJB Java Naming and Directory Interface (JNDI) names and `ejb-ref` resolutions, along with a list of application servers where the EJB JAR runs.

The product creates a composition unit from only one asset. However, multiple composition units can share a single asset. This is particularly useful in scenarios where different configurations use the same application binary files to provide different runtime behavior.

The following rules apply to a composition unit:

- A composition unit can exist only in a business-level application.
- Because a composition unit contains application-specific configuration and wiring information, multiple business-level applications cannot share an asset or shared library composition unit.

The following graphic shows the use of composition units in business-level applications. Assume that you have unprocessed files, such as archives, that you want to use in business-level applications. Before you can add the files to business-level applications, you must first import the files as assets, which adds the files to the product repository. Next, you add the assets to business-level applications, which creates composition units for the assets. Business-level applications can contain asset composition units, shared library composition units, or business-level composition units.



Importing assets

You must register application business logic such as Java Platform, Enterprise Edition (Java EE) archives, libraries, and other resource files with the product configuration as assets before you can add the assets to one or more business-level applications. Importing an asset registers it with the product configuration.

Before you begin

This topic assumes that you have one or more application binary files that you want to add to a business-level application. You must register those binary files as assets before you can add them to the business-level application.

About this task

Before a business-level application that uses an asset can be started on the target run time, the asset binaries must be extracted to a deployer-defined location on the file system that is local to the target run time. Importing an asset extracts binaries to a location that is local to the target run time.

The application server run time that reads the asset binaries either at application start time or while serving an incoming client request determines the extraction format of the asset binaries. The extraction format might include unzipping of Java archive (JAR) or compressed (zip) files.

This topic describes how to import an asset using the administrative console. Alternatively, you can use the wsadmin tool or programming.

1. Click **Applications** → **New Application** → **New Asset** in the console navigation tree.

2. On the Upload asset page, specify the asset package to import.

- a. Specify the full path name of the asset.
- b. Click **Next**.

3. On the Select options for importing an asset panel, specify asset settings.

You typically can click **Next** and use the default values.

a. Optional: For **Asset description**, specify a brief description of the asset.

b. Optional: For **Asset binaries destination URL**, specify the target location of the asset.

This setting specifies the location to which the product extracts the asset. After an asset is imported, the product looks for the asset in this location when a running application uses the asset.

If you do not specify a value, the product installs the asset to the default location,

`${profile_root}/installedAssets/asset_name/BASE/`.

c. Optional: For **Asset type aspects**, examine the asset content type and version specified by the product. You cannot change this setting value.

The type aspect typically denotes the type of application contents, such as a specification to which the application is written. For example, an enterprise bean (EJB) that supports the EJB Version 2.0 specification has the aspects `type=EJB,version=2.0`.

If the type aspect is none and if the asset is a JAR file, then the product associates a `javarchive` type aspect with the asset by default.

d. For **File permissions**, specify any file permissions that are set on asset binary files so the target run time can read or run the asset. Importing the asset extracts its binary files on the disk local to the target runtime environment.

Try importing the asset using the default value. For detailed information on the **File permissions** setting, refer to the Select options for importing an asset panel online help.

e. For **Current asset relationships**, add assets that the asset you are importing needs to run or remove assets that are not needed.

When the product imports a JAR asset, the product detects asset relationships automatically by matching the dependencies defined in the JAR manifest with the assets that are already imported into the administrative domain.

f. For **Validate asset**, specify whether the product validates the asset.

The setting is deselected by default. This **false (no)** value is appropriate for most assets. Only select **true (yes)** to validate an asset when needed.

The product does not save the value specified for **Validate asset**. Thus, if you select to validate the asset (**yes**) now and later update the asset, when you update the asset you must enable this setting again for the product to validate the updated files.

g. Click **Next**.

4. On the Summary page, click **Finish**.

Results

Several messages are displayed, indicating whether your asset is imported successfully.

An asset can contain multiple deployable objects as defined by the application contents of that asset. A *deployable object* is a part of the asset that you can map to a deployment target such as an application server. If the product imports the asset successfully, then appropriate deployable objects are identified in the asset and are further used when a composition unit is created from that asset.

If the asset importing is not successful, read the messages and try importing the asset again. Correct the values noted in the messages.

What to do next

If the product imports the asset successfully and displays the list of assets on the Assets page, then click **Save**.

Add a composition unit to a business-level application using the asset that you imported. An asset included in a business-level application is represented by a composition unit.

Upload asset settings

Use this panel to specify the asset to register with the asset repository. You can add registered assets to a business-level application.

To view this administrative console panel, click **Applications** → **New application** → **New Asset**.

Importing an asset registers the asset with the asset repository.

The product manages the contents of a registered asset as a single entity. The contents of a registered asset must be accessible to application servers, Web servers and other runtime environments that use the asset.

During asset importing, asset files typically are uploaded from a client workstation running the browser to the server running the administrative console, where they are registered. In such cases, use the Web browser running the administrative console to select files to upload to the server.

Path to the asset

Specifies the fully qualified path to the asset.

Specify one of the following supported assets:

- A single file, such as an enterprise bean (EJB) file
- An archive of files, such as a Java archive (JAR) or a compressed (zip) file
- An archive of archives, such as an enterprise archive (EAR) or shared library file

Use **Local file system** if the browser and asset files are on the same machine (whether or not the server is on that machine, too).

Use **Remote file system** if the asset file resides on any node in the current cell context. Only supported assets are shown during the browsing. Also use **Remote file system** to specify an asset file that is already residing on the machine running the application server. For example, the field value might be *profile_root/installableApps/my_bean.ejb*. After the asset file is transferred, the **Remote file system** value shows the path of the temporary location on the server.

Asset settings

Use this page to specify options for the registration of an asset with the asset repository. Default values for the options are used if you do not specify a value.

To view this administrative console page, click **Applications** → **Application Types** → **Assets** → **asset_name**. This page is similar to the Select options for importing an asset panel on the asset import and update wizards.

Asset name

Specifies a logical name for the asset. An asset name must be unique within a cell and cannot contain an unsupported character.

An asset name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

This **Asset name** field is the same as the **Name** setting on an Assets page.

Data type String

Asset description

Specifies a description for the asset.

Asset binaries destination URL

Specifies the directory to which the product imports the asset file.

Data type String
Units Full path name

Asset type aspects

Specifies the type of asset content. Examples of asset type include Java archive (JAR) files, shared libraries, and enterprise application archive (EAR) files.

The asset type suggests the content of the asset. For example, an asset packaged as a JAR file might contain a Web module, portlet and Web service.

This setting is read-only. You cannot edit this setting.

Data type String
Units File type
Default none

File permissions

Specifies access permissions for asset binaries that the product expands to the asset binaries destination URL.

You can specify file permissions in the text field. You can also set some of the commonly used file permissions by selecting them from the list. List selections overwrite file permissions set in the text field.

You can set one or more of the following file permission strings in the list. Selecting multiple options combines the file permission strings.

List option	File permission string set
Allow all files to be read but not written to	.*=755
Allow executables to execute	.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
Allow HTML and image files to be read by everyone	.*\.htm=755#.*\.html=755#.*\.gif=755#.*\.jpg=755

Instead of using the list to specify file permissions, you can specify a file permission string in the text field. File permissions use a string that has the following format:

file_name_pattern=permission#file_name_pattern=permission

where *file_name_pattern* is a regular expression file name filter (for example, *.*\.jsp* for all JSP files), *permission* provides the file access control lists (ACLs), and # is the separator between multiple entries of *file_name_pattern* and *permission*. If # is a character in a *file_name_pattern* string, use \# instead.

If multiple file name patterns and file permissions in the string match a uniform resource identifier (URI) within the asset, then the product uses the most stringent applicable file permission for the file. For example, if the file permission string is *.*\.jsp=775#a.*\.jsp=754*, then the *abc.jsp* file has file permission 754.

Note: Using regular expressions for file matching pattern compares an entire string URI against the specified file permission pattern. You must provide more precise matching patterns using regular expressions as defined by Java programming API. For example, suppose the product processes the following directory and file URIs during a file permission operation:

1	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war
2	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp
3	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/META-INF/MANIFEST.MF
4	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/WEB-INF/classes/MyClass.class
5	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/mydir/MyClass2.class
6	/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/META-INF

The file pattern matching results are:

- MyWarModule.war does not match any of the URIs
- .*MyWarModule.war.* matches all URIs
- .*MyWarModule.war\$ matches only URI 1
- .*\.jsp=755 matches only URI 2
- .*META-INF.* matches URIs 3 and 6
- .*MyWarModule.war/.*/.*\.class matches URIs 4 and 5

If you specify a directory name pattern for **File permissions**, then the directory permission is set based on the value specified. Otherwise, the **File permissions** value set on the directory is the same as its parent. For example, suppose you have the following file and directory structure:

```
/opt/WebSphere/profiles/AppSrv01/installedApps/MyCell/MyApp.ear/MyWarModule.war/MyJsp.jsp
```

and you specify the following file pattern string:

```
.*MyApp.ear$=755#.*\.*.jsp=644
```

The file pattern matching results are:

- Directory MyApp.ear is set to 755
- Directory MyWarModule.war is set to 755
- Directory MyWarModule.war is set to 755

Note: Regardless of the operation system, always use a forward slash (/) as a file path separator in file patterns.

Access permissions specified here are at the asset level. You can also specify access permissions for asset binaries in the node-level configuration. The node-level file permissions specify the maximum (most lenient) permissions that can be given to asset binaries. Access permissions specified here at asset level can only be the same as or more restrictive than those specified at the node level.

Data type String

Current asset relationships

Specifies the assets to which this asset is related.

To add or remove a relationship, use the Manage relationships panel:

1. Click **Manage Relationships** to access the Manage relationships panel. The **Selected** list on the right lists the current asset relationships.
2. To add a relationship, select an asset in the **Available** list on the left and click >>.
3. To remove a relationship, select an asset in the **Selected** list on the right and click <<.
4. Click **OK**.

Data type String

Default none

Validate asset

Specifies whether the product examines the asset references specified during asset importing or updating and, if validation is enabled, warns you of incorrect references or fails the operation.

An asset typically refers to resources using data sources for container-managed persistence (CMP) beans or using resource references or resource environment references defined in deployment descriptors. The validation checks whether the resource referred to by the asset is defined in the scope of the deployment target of that asset.

Select `true` (enable the check box) for resource validation and to stop operations that fail as a result of incorrect resource references. Select `false` (empty check box) for no resource validation.

Data type String

Default false (empty check box)

Managing assets

After application binary files are imported and registered with the product management domain as assets, you can view, update and export those assets.

Before you begin

Import one or more assets. The name of each imported assets is shown on the list of assets on the administrative console Assets page.

About this task

You can view the contents of assets, update assets, remove assets from the product management domain, or export copies of assets to a target location. This topic describes how to perform these asset management operations from the administrative console Assets page. Alternatively, you can use programming or the wsadmin tool.

- View or edit asset settings.
 1. Go to the administrative console Assets page.
Click **Applications** → **Application Types** → **Assets**.
 2. Click the asset name in the list of assets. The Asset settings page displays the values that are specified for the asset.
 3. Optional: Change the asset settings as needed and click **OK** to save the changes.
- Remove one or more assets from the product management domain.
- Update the contents of an asset.
- Export an asset to a target location.

What to do next

Create a business level application and add the asset to the business-level application.

Asset collection

Use this page to view a list of assets in the asset repository and to manage those assets. After importing an asset, you can add the asset to a business-level application.

Assets include Java archive (JAR) and compressed files that are used by applications installed on a server.

To view this administrative console page, click **Applications** → **Application Types** → **Assets**.

To view the values specified for an asset, click the asset name in the list. The displayed asset settings page shows the values specified. On the settings page, you can change existing asset values.

To manage an asset, enable the **Select** check box beside the asset name in the list and click a button:

Button	Resulting action
Import	Opens a wizard that helps you add an asset to the asset repository.
Delete	Removes the asset from the asset repository and deletes the asset binaries from the file system of all nodes where the assets are installed. On single-server installations, deletion occurs after the configuration is saved.

Button	Resulting action
Update	Opens a wizard that helps you update asset files. You can replace a file or module that exists on the server with a file or module that has the same name. Or you can add a new file or module, provided the new file or module does not have the same name as an asset that already exists on the server.
Export	Accesses the Export asset page, which you use to export an asset to a file at a location of your choice. Use the Export action to back up an asset.

Name

Specifies the name of the asset. Asset names must be unique within a cell and cannot contain an unsupported character.

Description

Specifies a description for the asset.

Updating assets

You can update application binary files that are assets registered with the product management domain.

Before you begin

Import one or more assets. The file name of each deployable object in the imported assets is shown on the list of assets on the administrative console Assets page.

About this task

You can update all of part of the contents of assets that are in the product management domain. This topic describes how to update an asset using the administrative console Update asset wizard. Alternatively, you can update assets using programming or the wsadmin tool.

1. Go to the Update asset wizard.
 - a. Click **Applications** → **Application Types** → **Assets** to access the Assets page.
 - b. Select the check box beside the asset that you want to update.
 - c. Click **Update**.
2. On the Update asset panel, specify whether you want replace an entire asset or update its contents and, as needed, the replacement file or module.
 - a. Select an update option.

You can update asset contents by adding, deleting, or updating a single file or module in the asset, or by merging multiple files or modules. Update options include the following:

 - Replace entire asset
 - Replace specific asset contents
 - Add module or file to asset
 - Remove file or module from asset
 - Merge asset contents

The online help for the Update asset panel describes the options.
 - b. If you are updating specific asset contents or removing a file or module, specify the path beginning with the asset archive file.

For **Specify the path beginning with the asset archive file**, specify a relative path to the file that starts from the root of the asset file. For example, if the file is located at com/company/greeting.class in module hello.jar, specify a relative path of hello.jar/com/company/greeting.class.
 - c. If you are updating the entire asset, updating an asset file or module, or merging asset contents, specify the full path name of the new file or module.
 - d. Click **Next**.

3. On the Select options for updating an asset page, specify asset settings and click **Next**.
The online help for the Select options for importing an asset panel describes the settings.
4. On the Summary page, click **Finish**.

Results

If you update an asset packaged as a library JAR file that is not a Java Platform, Enterprise Edition (Java EE) archive, then the product automatically distributes the updated asset to all of the composition units that use the asset.

However, if you update a Java EE asset, then the product does not automatically distribute the updated Java EE archive to composition units created from that asset, which are Java EE applications. You must select every Java EE application created from that asset and use the **Update** button to update the Java EE application individually by specifying the update contents.

What to do next

Create a business-level application and add the asset to the business-level application.

Update asset settings

Use this panel to select whether you want replace an entire asset or update its contents. You can update asset contents by adding, deleting, or updating a single file or module in the asset, or by merging multiple files or modules into an asset. Updating an asset registers the updated files with the product management domain.

To view this administrative console panel, click **Applications** → **Application Types** → **Assets**, select the asset to update, and then click **Update**.

The product manages the contents of a registered asset as a single entity. The contents of a registered asset must be accessible to application servers, Web servers and other runtime environments that use the asset.

When you replace an asset or update an asset by adding a file or module, asset files typically are uploaded from a client workstation running the browser to the server machine running the administrative console, where they are registered. In such cases, use the Web browser running the administrative console to select files to upload to the server machine.

The specified asset that you are installing must be one of the following supported assets:

- A single file, such as an enterprise bean (EJB) file
- An archive of files, such as a Java archive (JAR) or a compressed (zip) file
- An archive of archives, such as an enterprise archive (EAR) or shared library file

Replace entire asset:

Under **Select the type of update to perform**, specifies to replace the entire asset installed on the server with a new (updated) asset.

After selecting this option, specify whether the asset is on a local or remote file system and the full path name of the asset. The path provides the location of the updated asset before installation.

Use **Local file system** if the browser and asset files are on the same machine (whether or not the server is on that machine, too).

Use **Remote file system** if the asset file resides on any node in the current cell context. Only supported assets are shown during the browsing. Also use **Remote file system** to specify an asset file that is already residing on the machine running the application server. For example, the field value might be

`profile_root/installableApps/my_bean.ejb`. After the asset file is transferred, the **Remote file system** value shows the path of the temporary location on the server.

Replace specific asset contents:

Under **Select the type of update to perform**, specifies to replace a file or module of the asset installed on the server.

After selecting this option, do the following:

1. For **Specify the path beginning with the asset archive file**, specify a relative path to the file that starts from the root of the asset file. For example, if the file is located at `com/company/greeting.class` in module `hello.jar`, specify a relative path of `hello.jar/com/company/greeting.class`.
2. Specify whether the asset is on a local or remote file system and the full path name of the asset. The path provides the location of the updated asset before installation.
3. Click **Next**.

The **Replace entire asset** description describes options for specifying the full path name of an asset or file to add using **Local file system** and **Remote file system** options.

Add a module or file to an asset:

Under **Select the type of update to perform**, specifies to add a file to the asset installed on the server.

After selecting this option, do the following:

1. For **Specify the path beginning with the asset archive file**, specify a relative path to the file that starts from the root of the asset file. For example, if the file is located at `com/company/greeting.class` in module `hello.jar`, specify a relative path of `hello.jar/com/company/greeting.class`.
2. Specify whether the asset is on a local or remote file system and the full path name of the asset. The path provides the location of the updated asset before installation.

The **Replace entire asset** description describes options for specifying the full path name of an asset or file to add using **Local file system** and **Remote file system** options.

Remove a file or module from an asset:

Under **Select the type of update to perform**, specifies to remove a file or module from the asset installed on the server.

After selecting this option, do the following:

1. For **Specify the path beginning with the asset archive file**, specify a relative path to the file to be removed that starts from the root of the asset file. For example, if the file is located at `com/company/greeting.class` in module `hello.jar`, specify a relative path of `hello.jar/com/company/greeting.class`.
2. Click **Next**.

Merge asset contents:

Under **Select the type of update to perform**, specifies to compare the new file or module with the file or module of the asset installed on the server. If the file or module exists, it is replaced. Otherwise, it is added to the installed asset.

After selecting this option, specify whether the new file or module is on a local or remote file system and the full path name of the file or module. The path provides the location of the updated asset before installation.

The **Replace entire asset** description describes options for specifying the full path name of a file or module to merge using **Local file system** and **Remote file system** options.

Deleting assets

You can remove application binary files that are registered as assets from the product management domain.

Before you begin

Import one or more assets. The name of each imported asset is shown on the list of assets on the administrative console Assets page.

About this task

You can remove assets from the product management domain, provided the asset does not have an existing composition unit. If an asset has one or more composition units defined in the management domain, then you cannot delete that asset until those composition units are removed.

This topic describes how to delete assets using the administrative console. Alternatively, you can use programming or the wsadmin tool.

1. Go to the Delete asset page.
 - a. Click **Applications** → **Application Types** → **Assets** to access the Assets page.
 - b. Select the check box beside the asset that you want to delete.
 - c. Click **Delete**.
2. On the Delete asset page, click **OK** to confirm that you want the specified asset removed from the product management domain.
Click **Cancel** to return to the Assets page and not delete the asset.

Results

The product deletes the asset from the product management domain.

What to do next

On the Assets page, verify that the deleted asset is no longer in the list of imported assets.

Exporting assets

After application binary files are imported and registered with the product management domain as assets, you can export those assets.

Before you begin

Import one or more assets. The file name of each deployable object in the imported assets is shown on the list of assets on the administrative console Assets page.

About this task

You can export copies of assets to a target location. Exporting stores application binary files, enabling you to back up the files or edit them. The file resulting from exporting an asset contains configuration information for the asset.

This topic describes how to export an asset from the administrative console Assets page. Alternatively, you can use programming or the wsadmin tool.

1. Go to the Export asset page.
 - a. Click **Applications** → **Application Types** → **Assets** to access the Assets page.
 - b. Select the check box beside the asset that you want to export.
 - c. Click **Export**.
2. On the Export asset page, click the asset name or identifier.
To cancel the export operation and return to the Assets page, click **Back**.
3. Specify the target location for the asset file.

What to do next

Examine the target file to verify that the asset exported correctly. You can later edit this file and import the edited asset.

Creating business-level applications

You can create an empty business-level application and then add assets, shared libraries, business-level applications, and other artifacts as composition units to the empty business-level application.

Before you begin

Configure each target application server as needed. You must deploy a business-level application to a Version 7.0 server.

Optionally, determine what assets or other files that you want to add to your business-level application and whether your application files can run on your deployment targets.

About this task

You can create business-level applications using the administrative console, programming, or the wsadmin tool.

1. Select a way to create your business level application.

Table 8. Ways to create business level applications

Option	Method
Administrative console business-level application creation wizard See “Creating business-level applications with the console” on page 375.	Click Applications → New application → New Business-level Application and follow instructions in the wizard.
Administrative console Java Platform, Enterprise Edition (Java EE) application installation wizard See “Installing enterprise application files with the console” on page 238.	Click Applications → New application → New Enterprise Application and follow instructions in the wizard. The product creates a new business-level application with the enterprise application that you install or makes the enterprise application a composition unit of an existing business-level application. See the Business-level application name setting on the Select installation options wizard panel.

2. Create your business-level application using the administrative console, programming or wsadmin.
3. Save the changes to your administrative configuration.

Results

The name of the application is shown in the list on the Business-level applications page.

What to do next

After you create a business-level application, you can do the following to add composition units to it:

1. Import any assets needed by your business-level application.
2. Add assets, shared libraries, or other business-level applications as composition units.
3. Save the changes to your administrative configuration.
4. Start the business-level application.

If the application does not run as desired, edit the application configuration, then save and run it again.

Creating business-level applications with the console

You can create an empty business-level application and then add assets or business-level applications as composition units to the empty business-level application.

Before you begin

Before you create a business-level application, decide upon an application name. Optionally, determine which assets, shared libraries, or business-level applications that the new business-level application needs.

About this task

This topic describes how to create an empty business-level application and then add assets as composition units to the application using the administrative console. Alternatively, you can use programming or the wsadmin tool.

You can add an asset or shared library composition unit to multiple business-level applications. However, each composition unit for the same asset must have a unique composition unit name. You can add a business-level application composition unit to more than one business-level application.

1. Create an empty business-level application.
 - a. Click **Applications** → **New application** → **New Business Level Application**.
 - b. On the New business-level application panel, specify a unique name for the application and a description, and then click **OK**.
 - c. On the business-level application settings page, click **Save**.

The name and description are shown in the list of applications on the Business-level applications page. Because the application is empty, its status is `Unavailable`.

2. Optional: Add one or more assets, non-Java EE shared libraries, or business-level applications to a business-level application. The product adds these assets as composition units of your business-level application.

If the asset that you want to add to your business-level application is a Java Platform, Enterprise Edition (Java EE) application or module that is not yet deployed, see step 3. If the asset is a Java EE shared library, see step 4.

- a. Import the assets or create the business-level applications that you want to add to the business-level application.

- b. Go to the business-level application settings page.

Click **Applications** → **Application Types** → **Business-level applications** → *application_name*.

- c. On the business-level application settings page, specify the type of composition unit to add.

- To add an asset, under **Deployed assets**, click **Add** → **Add Asset**.
 - To add a shared library, under **Deployed assets**, click **Add** → **Add Shared Library**.
 - To add a business-level application, under **Business-level applications**, click **Add**.
- d. On the Add panel, select a unit from the list of available units, and then click **Continue**.
If you are adding one or more deployable unit assets and you have multiple imported assets available, you can select more than one deployable unit.
 - e. On the Set options panel, change the composition unit settings as needed, and then click **Next**.
This panel is not shown when you add a Java EE asset as a shared library or if you have multiple deployable unit assets. If the application installation or update wizard displays and you want to add a Java EE asset as a shared library, see step 4.
 - f. On the Map composition unit to a target panel, change the deployment target as needed, and then click **Next**.
This panel is not shown when you add a business-level application.
 - g. If you are adding one or more deployable unit assets, specify composition unit relationship options.
See “Deployable unit relationship settings” on page 383.
 - h. On the Summary page, click **Finish**. Several messages are displayed, indicating whether the product adds the unit to the business-level application successfully. A message having the format `Completed res=[WebSphere:cuname=unit_name,cuedition=version]` indicates that the addition is successful. Click **Manage application**.
If the product adds the unit successfully, the name of the unit is shown on the list of composition units on the Adding composition unit to the business-level application page.
If the unit addition is not successful, read the messages and try adding the unit again. Correct the problems noted in the messages.
 - i. On the Adding composition unit to the business-level application page, click **Save**.

The product creates composition units for the asset, shared library, or business-level application. The unit names are shown in lists of composition units on the settings page of your business-level application. To view the settings page, click **Applications** → **Application Types** → **Business-level applications** → *your_application_name*.

3. Optional: Install a Java EE application or module, and add it as a composition unit to your business-level application.

When installing an enterprise archive (EAR) file or a stand-alone Java EE module using the application installation wizard, you can specify a business-level application to which to add the EAR file or module. You can also specify relationships to any shared libraries that your Java EE application or module uses. The product creates composition units that represent those relationships.

- a. Click **Applications** → **New application** → **New Enterprise Application**.
- b. On the first Preparing for the application installation panel, specify the Java EE application or module to install and click **Next**.
- c. On the second Preparing for the application installation panel, select **Detailed - Show all installation options and parameters**, specify whether to generate default bindings and mappings as needed for the application or module, and click **Next**.
- d. On the Select installation options panel of the wizard, select your business-level application for **Business-level application name** and click **Next**. The product creates a composition unit that has the same name as the Java EE application or module and adds the unit to your business-level application.

If you do not specify a value for **Business-level application name**, then the product creates a default business-level application that has the same name as the Java EE application that you are installing. The product does not add the Java EE application as a composition unit to the business-level application that you created in step 1.

- e. Optional: On the Map shared library relationship panel of the wizard, specify relationship identifiers and composition unit names for shared libraries that modules in your Java EE application use. The product creates a composition unit for each shared library relationship in your business-level application.

You can map shared library relationships when installing your Java EE application or module or, after installation, return to the Map shared library relationship panel and specify shared library relationships. See step 4.

- f. Complete the other application installation wizard options as needed to install the Java EE application or module.

The product creates composition units for the application, module, or shared library relationships. The unit names are shown in lists of composition units on the settings page of your business-level application. To view the settings page, click **Applications** → **Application Types** → **Business-level applications** → *your_application_name*.

4. Optional: After installation of a Java EE application or module, specify composition units for relationships to shared libraries used by your business-level application on the Map shared library relationship panel of the application installation or update wizard.
 - a. If you have not done so already, import a Java EE asset such as an enterprise bean (EJB) or Web module (WAR) that uses a shared library file.

If the product displays `javaarchive` for **Asset type aspects** on the asset settings page, continue to step 4b.

If the product does not display `javaarchive` for **Asset type aspects** on the asset settings page, then the asset is not a Java EE asset. Use step 2 to add a shared library to your business-level application.
 - b. Go to a settings page for your business-level application.

Click **Applications** → **Application Types** → **Business-level applications** → *your_application_name*.
 - c. Under **Deployed assets**, click **Add** → **Add Shared Library**.
 - d. On the Add composition unit panel, select the Java EE asset that you imported and then click **Continue**.

The Java EE application installation or update wizard displays. Select the Java EE application or module that uses the asset, and complete the steps in the wizard.
 - e. On the Select installation options panel of the wizard, select your business-level application for **Business-level application name**.
 - f. On the Map shared library relationship panel of the wizard, specify a relationship identifier and composition unit name for the asset.
 - g. Complete the other wizard options as needed.

The product creates a composition unit for the shared library relationship. The unit name is shown in the list of deployed asset composition units on the settings page of your business-level application.

Results

The name of your business-level application is shown on the Business-level applications page in the list of applications.

What to do next

After you create the application, save the changes to your configuration and start the application as needed.

Business-level application collection

Use this page to view and manage business-level applications.

To view this administrative console page, click **Applications** → **Application Types** → **Business-level applications**.

To view the values specified for an application configuration, click the application name in the list. The displayed application settings page shows the values specified. On the settings page, you can change existing configuration values and link to additional console pages that assist you in configuring the application.

To manage a business-level application, enable the **Select** check box beside the application name in the list and click a button:

Button	Resulting action
Start	Attempts to run the application. After the application starts successfully, the state of the application changes to <i>Started</i> if the application starts on all deployment targets, else the state changes to <i>Partial Start</i> .
Stop	Attempts to stop the processing of the application. After the application stops successfully, the state of the application changes to <i>Stopped</i> if the application stops on all deployment targets, else the state changes to <i>Partial Stop</i> .
New	Opens a wizard that helps you add assets, shared libraries, or business-level applications as composition units to your application.
Delete	Deletes the application from the product configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed. On single-server installations, deletion occurs after the configuration is saved.

Name:







Specifies the name of the business-level application. Application names must be unique within a cell and cannot contain an unsupported character.

Description:

Specifies a description for the business-level application.

Status:

Indicates whether the application deployed on the application server is started, stopped, or unknown.

	Started	Application is running.
	Partial start	Application is in the process of changing from a <i>Stopped</i> state to a <i>Started</i> state. Application is starting to run but is not fully running yet. Or, it cannot fully start because a server mapped to one or more application modules is stopped.
	Stopped	Application is not running.
	Partial stop	Application is in the process of changing from a <i>Started</i> state to a <i>Stopped</i> state. Application has not stopped running yet.
	Unknown	Status cannot be determined.
	Pending	Status is temporarily unknown pending an event that a user did not initiate, such as pending an asynchronous call.
	Not applicable	Application does not provide information as to whether it is running.

The status of an application on a Web server is always **Unknown**.

New business-level application settings

Use this panel to name and describe a new business-level application.

To view this administrative console panel, click **Applications** → **New application** → **New Business-level Application**.

Name:

Specifies a logical name for the business-level application. An application name must be unique within a cell and cannot contain an unsupported character.

An application name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

Data type String

Description:

Specifies a description for the application.

This field is the same as the **Description** setting on a Business-level applications page.

Shared library relationship and mapping settings

Use the Shared library relationship and Shared library relationship mapping pages to specify relationship identifiers and composition unit names for shared libraries that modules in your enterprise application reference. When installing your enterprise application, the product creates a composition unit for each shared library relationship in the business-level application that you specified on the Select installation options panel of the application installation wizard.

To view this console panel in a wizard, click **Applications** → **Install new application** → **New Enterprise Application** → *application_path* → **Next** → **Detailed - Show all installation options and parameters** → **Next** → *application_name* → **Step: Map shared library relationships**.

After installation, click **Applications** → **Application Types** → **WebSphere enterprise applications** → **Shared library relationships**

To map library files used in a business-level application with an application or Web module, use the Shared library relationship mapping page:

1. Click **Reference shared libraries**.
2. Note the application or module in **Map libraries to the application or module listed**. You are associating library files with that application or module.
3. From the **Available** list, select one or more libraries that the application or module uses.

4. Click >> to add them to the **Selected** list.
5. To remove an association, select one or more libraries in the **Selected** list and click <<.
6. Click **OK**.

Module:

Specifies the name of the module associated with the shared libraries.

URI:

Specifies the location of the module relative to the root of the application EAR file.

Relationship identifiers:

Specifies an identifier for a module shared library relationship. The product assigns an identifier to the composition unit that it creates for the shared library relationship in the business-level application.

Composition unit names:

Specifies a composition unit name for the shared library relationship. The product uses this value to name the composition unit that it creates for the shared library relationship in the business-level application that you specified on the Select installation options panel of this wizard.

This setting is only in the application installation and update wizards.

Match target:

Specifies whether the product maps the composition unit for the shared library relationship to the same deployment target as the business-level application.

Add composition unit settings

Use this panel to specify options for the composition unit to be added to the business-level application. The product assigns a default value for an option when you do not specify a value.

To view this administrative console panel, click **Applications** → **Application Types** → **Business-level applications** → *business-level_application_name* → **Add** → **Add unit_type**.

Name:

Specifies the name of the composition unit to be added to the business-level application.

The table lists available composition units. Select a unit from this list.

Description:

Specifies a description for the composition unit.

Add asset settings

Use this panel to add one or more assets to a business-level application.

To view this administrative console panel, click **Applications** → **Application Types** → **Business-level applications** → *application_name* → **Add** → **Add Asset**.

Deployable units:

Specifies the imported assets available for use in a business-level application. The list of deployable units includes only imported assets, and not shared libraries or business-level applications.

From this list, select one or more deployable units to add as composition units to your business-level application.

Set options settings

Use this panel to specify options for the composition unit to be added to the business-level application. The product supplies default values for the options if you do not specify a value.

To view this administrative console panel, click **Applications** → **Application Types** → **Business-level applications** → *application_name*. On the business-level application settings page, specify the type of composition unit to add:

- To add an asset, under **Deployed assets**, click **Add** → **Add Asset**.
- To add a shared library, under **Deployed assets**, click **Add** → **Add Shared Library**.
- To add a business-level application, under **Business-level applications**, click **Add**.

Backing identifier:

Specifies a unique identifier for a composition unit that is registered in the application management domain.

The identifier has the format: `WebSphere:unit_typename=unit_name,unit_typeversion=version_number`. For example, for the MyApp.jar asset, the backing identifier might be `WebSphere:assetname=MyApp.jar`.

Data type String
Units Composition unit identifier

Name:

Specifies the name of the composition unit.

For example, for the MyApp.jar asset, the name might be MyApp.jar.

A unit name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

Data type String

Description:

Specifies a description for the composition unit.

Starting weight:

Specifies the order in which composition units are started when the server starts. The starting weight is like the startup order. The composition unit with the lowest starting weight is started first.

The value that you set for **Starting weight** determines the importance or weight of a composition unit within the business-level application. For example, for the most important composition unit within a business-level application, specify 1 for **Starting weight**. For the next most important composition unit within the business-level application, specify 2 for **Starting weight**, and so on.

Data type	Integer
Default	1
Range	0 to 2147483647

Start composition unit upon distribution:

Specifies whether to start the composition unit after the product distributes the composition unit to other locations.

The default is not to start the composition unit.

Data type	Boolean
Default	false

Restart behavior on update:

Specifies whether the product restarts deployment targets after updates to the composition unit.

Usually, a composition unit is mapped to one or more deployment targets. This setting determines whether the product restarts those targets after editing the composition unit.

Option	Description
ALL	The product restarts each target node of the composition unit after editing the composition unit.
DEFAULT	The product restarts the nodes controlled by the sync plug-ins after editing the composition unit.
NONE	The product does not restart nodes after editing the composition unit.

Map target settings

Use this panel to map a composition unit to a deployment target. The product assigns a default target when you do not specify a target.

To view this administrative console panel, click **Applications** → **Application Types** → **Business-level applications** → **application_name** → **composition_unit_name** → **Modify Target**. The Map target page is similar to the Map composition unit to a target panel in the add composition unit wizard.

On single-server products, a deployment target can be an application server or Web server.

On this panel, map a composition unit to one or more desired targets.

Current targets:

Specifies the existing deployment targets for the composition unit.

Available:

Lists the names of available deployment targets. This list is the same for every composition unit that is registered in the cell.

From this list, select only appropriate deployment targets for a composition unit.

If the unit calls a Version 7.x application programming interface (API) or uses a 7.x feature, then you must map the unit to a 7.x deployment target. If the unit supports Java Platform, Enterprise Edition (Java EE) 5, then you must map the unit to a 7.x deployment target. If the unit supports Java 2 Platform, Enterprise Edition (J2EE) 1.4, then you must map the unit to a 6.x or 7.x deployment target. You can map units that call a 6.x API or use a 6.x feature to a 6.x or 7.x deployment target.

To map a composition unit to a deployment target, select a target from the **Available** list and click >>. The target name is displayed in the **Selected** list.

Selected:

Lists the names of desired deployment targets.

When you click **OK**, the product maps the composition unit to the deployment targets in the **Selected** list.

To remove a deployment target from the **Selected** list, select the target and click <<.

Deployable unit relationship settings

Use this panel to specify relationship options for deployable units in an asset deployed as part of a business-level application. Specifying a relationship declares a dependency relationship that a deployable unit has on another asset deployed as a shared library in the same business-level application.

To view this administrative console panel, click **Applications** → **Application Types** → **Business-level applications** → *application_name* → *deployed_asset_name* → **Manage Relationships**. This help also pertains to both the Relationship options panel and the Composition unit relationship options panel that are shown when you add multiple deployable unit assets to a business-level application. These panels are shown for the **Define relationship with existing composition units** and **Options for creating new composition units to satisfy asset relationships** wizard steps.

A business-level application consists of composition units. When you add an asset to a business-level application, the product creates a composition unit for the asset. The composition unit name can be different from the name of the asset being deployed. The list of deployed assets shown for a business-level application consists of the composition unit names for the deployed assets. The relationships defined in this panel are composition unit relationships. The deployable units listed for a composition unit are those you chose from the associated asset when adding the asset. Composition unit relationships are expressed as deployable unit dependencies on other composition units belonging to the same business-level application. Only a composition unit for an asset deployed as a shared library can be specified as a dependency. You can map each deployable unit to a target independently from the others. Modifying relationships in this panel only affects the composition unit, not the associated asset.

To specify relationship options, select a deployable unit and click a button.

Button	Resulting action
Set Relationships	Displays a panel through which you can add or change relationships for the deployable unit. Specify a relationship if a deployable unit depends on another asset deployed as a shared library in order to run. This button is on the Set relationship options panel.
Enable Match Targets	If the deployable unit has a dependency relationship defined, click Enable Match Targets to map the related deployed assets to the same deployment targets as the dependent deployable unit.

Button	Resulting action
Disable Match Targets	If the deployable unit has a dependency relationship defined, click Disable Match Targets if the related deployed assets do not need to be deployed to the same targets as the deployable unit.

Deployable unit name:

Specifies the name of the deployable unit of the selected deployed asset.

Relationship:

Specifies the composition unit names for all relationships defined for the associated deployable unit.

This setting is on the Set relationship options panel.

By default, a deployable unit has no relationships. To add or change related composition units, do the following:

1. Select the deployable unit.
2. Click **Set Relationships**.
3. Select the composition units that the deployable unit requires by moving them from the **Available** list to the **Selected** list.
4. Click **OK**.

Match targets:

Indicates the match targets value selected for the associated deployable unit. The default value is true.

A match targets value of true maps the composition units listed under **Relationship** to the same deployment targets as the associated deployable unit. Typically, you must deploy related composition units to the same targets as the dependent deployable unit in order for the deployable unit to run.

A false value indicates that the related composition unit can map to deployment targets which are different from the deployment targets of the deployable unit.

To set the value to true, select the deployable unit and click **Enable Match Targets**. To set the value to false, select the deployable unit and click **Disable Match Targets**. To set this value, the deployable unit must have a related composition unit.

Business-level application settings

Use this page to configure a business-level application.

To view this administrative console page, click **Applications** → **Application Types** → **Business-level applications** → *application_name*.

This page is the same as the Adding composition unit to the business-level application page.

Name

Specifies a logical name for the application. An application name must be unique within a cell and cannot contain an unsupported character.

An application name cannot begin with a period (.), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Unsupported characters		
/ forward slash	\$ dollar sign	' single quote mark
\ backslash	= equal sign	" double quote mark
* asterisk	% percent sign	vertical bar
, comma	+ plus sign	< left angle bracket
: colon	@ at sign	> right angle bracket
; semi-colon	# hash mark	& ampersand (and sign)
? question mark]]> No specific name exists for this character combination	

Data type String

Description

Specifies a description for the business-level application.

Deployed assets

Specifies the asset and shared library composition units in the business-level application. A *composition unit* is a registered asset or shared library that has additional configuration information, which you specify when adding the asset to the application.

For each composition unit, the table provides a name, description, asset type, and the runtime status of the composition unit.

Button	Resulting action
Add > Add Asset	For assets that contain Java Platform, Enterprise Edition (Java EE) applications or modules, opens the application installation wizard. On the Select installation options panel of this wizard, you can specify a Business-level application name value that identifies the target business-level application. On the Map shared library relationships panel, you can identify the shared library files that individual modules need to run and specify composition unit names for the module-shared library relationships. For non-Java EE assets, opens a wizard that helps you add an asset as a composition unit to your business-level application.
Add > Add Shared Library	Opens a wizard that helps you add a library file as a composition unit to your business-level application.
Delete	Deletes the composition unit from the product configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed. On single-server installations, deletion occurs after the configuration is saved.

Business-level applications

Specifies the business-level applications in this business-level application.

The table provides a name, description, and the runtime status of each contained business-level application.

Button	Resulting action
Add	Opens a wizard that helps you add a business-level application to your business-level application.

Button	Resulting action
Delete	<p>Deletes the business-level application from the product configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed.</p> <p>On single-server installations, deletion occurs after the configuration is saved.</p>

Composition unit settings

Use this page to view composition unit settings and to change the deployment target of a composition unit.

To view this administrative console page, click **Applications** → **Application Types** → **Business-level applications** → *application_name* → *composition_unit_name*.

Name

Specifies a logical name for the composition unit. You cannot change the name on this page.

Description

Specifies a description for the composition unit. You cannot change the description on this page.

Backing identifier

Specifies a unique identifier for a composition unit that is registered in the application management domain.

The identifier has the following format: `WebSphere:unit_typename=unit_name`. For example, for the `MyApp.jar` asset, the backing identifier might be `WebSphere:assetname=MyApp.jar`.

You cannot change the identifier on this page.

Data type	String
Units	Configuration unit identifier

Starting weight

Specifies the order in which composition units are started when the server starts. The starting weight is like the startup order. The composition unit with the lowest starting weight is started first.

The value that you set for **Starting weight** determines the importance or weight of a composition unit within the business level application. For example, for the most important composition unit within a business-level application, specify 1 for **Starting weight**. For the next most important composition unit within the business-level application, specify 2 for **Starting weight**, and so on.

Data type	Integer
Default	1
Range	0 to 2147483647

Start on distribution

Specifies whether to start the composition unit when the product distributes the composition unit to other locations.

The default is not to start the composition unit.

This setting applies to asset or shared library composition units. This setting does not apply when the composition unit is a business-level application.

Data type	Boolean
------------------	---------

Default false

Recycle behavior on update

Specifies whether the product restarts the composition unit after the composition unit is updated.

The default is to restart the composition unit after partial updating of the composition unit.

This setting applies to asset or shared library composition units. This setting does not apply when the composition unit is a business-level application.

Option	Description
ALL	Restarts the composition unit after the entire composition unit is updated
DEFAULT	Restarts the composition unit after the part of the composition unit is updated
NONE	Does not restart the composition unit after the composition unit is updated

Current target

Specifies the existing deployment target for the composition unit.

To change the deployment target, click **Modify target** and select a different deployment target from the list of available clusters and servers.

Example: Creating a business-level application

You can add many different types of artifacts to business-level applications. For example, you can add Java Platform, Enterprise Edition (Java EE) applications or modules, Java archives (JAR files), data in compressed files, and other business-level applications.

About this task

An example of creating a simple business-level application follows. This example assumes that you have a compressed file, such as a zip file, or other archive available on your computer or on a remote server that you can use to complete the example.

If you do not have a compressed file available, look in product directories. Installing the product samples adds several sample files to the /samples directory. You can use these sample files in a business-level application.

1. Import assets.
 - a. Click **Applications** → **New application** → **New Asset** in the console navigation tree.
 - b. On the Upload asset page, specify the asset package to import and click **Next**.
For example, specify a compressed file such as a zip file and click **Next**.
 - c. On the Select options for importing an asset panel, click **Next**.
 - d.
 - e. On the Summary panel, click **Finish**.
 - f. On the Adding asset to repository panel, if messages show that the operation completed, click **Manage assets**.
 - g. On the Assets page, click **Save**.

The file name displays in the list of assets.

2. Create an empty business-level application named MySampleBLA.
 - a. Click **Applications** → **New application** → **New Business Level Application**.

- b. On the New business-level application panel, specify a unique name such as MySampleBLA and a description, and then click **OK**.
- c. On the business-level application settings page, click **Save**.

The name and description are shown in the list of applications on the Business-level applications page. Because the application is empty, its status is `Unavailable`.

3. Add the asset composition unit to your business-level application.
 - a. On the Business-level applications page, click the application name in the list of applications.
 - b. On the business-level application settings page, click **Add** → **Add Asset**.
 - c. On the Add composition unit panel, select an asset composition unit from the list of available units, and then click **Continue**.

For example, select the compressed file asset and then click **Continue**.
 - d. On the Set options panel, click **Next**.
 - e. On the Map composition unit to a target panel, change the target server as needed, and then click **Next**.
 - f. On the Summary panel, click **Finish**. Several messages are displayed. A message having the format `Completed res=[WebSphere:cuname=unit_name]` indicates that the addition is successful.
 - g. If the addition is successful, click **Manage application**.
 - h. On the business-level application settings page, click **Save**.

The asset name and type displays in the list of deployed assets.

4. Start the business-level application.
 - a. Click **Applications** → **Application Types** → **Business-level applications**.
 - b. On the Business-level applications page, select the check box beside your application.
 - c. Click **Start**.

When the business-level application is running, a green arrow displays for **Status**. If the business-level application does not start, ensure that the deployment target to which the application maps is running and try starting the application again.

What to do next

You can add other assets to your business-level application.

Starting business-level applications

You can start a business-level application that is not running (has a status of `Stopped`). The application must contain code that can run on a server to start.

Before you begin

The application must be installed on a server. By default, the application starts automatically when the server starts.

About this task

You can start and stop business-level applications manually using the administrative console or `wsadmin` commands.

This topic describes how to use the administrative console to start a business-level application.

1. Go to the Business-level applications page.

Click **Applications** → **Application Types** → **Business-level applications** in the console navigation tree.

2. Select the check box for the application you want started.
3. Click **Start**. The product runs the application and changes the state of the application to Started. The status is changed to partially started if not all servers on which the application is deployed are running.

Results

A message stating that the application started displays at the top the page.

What to do next

To restart a running application, select the application you want to restart, click **Stop** and then click **Start**.

Stopping business-level applications

You can stop a business-level application that is running and has a status of Started).

Before you begin

The application must be running on a product server.

About this task

You can stop applications manually using the administrative console or wsadmin commands.

This topic describes how to use the administrative console to stop a business-level application.

1. Go to the Business-level applications page.
Click **Applications** → **Application Types** → **Business-level applications** in the console navigation tree.
2. Select the check box for the application you want stopped.
3. Click **Stop**. The product stops the processing of the application and changes the state of the application to Stopped.

Results

The status of the application changes and a message stating that the application stopped displays at the top the page.

What to do next

To restart a stopped application, select the application you want to restart, and then click **Start**.

Updating business-level applications

You can update business-level applications by deleting or changing composition units, or by mapping composition units to different deployment targets.

Before you begin

Determine what changes that you want to make to your application. Also, determine whether the changed application can run on your deployment targets.

About this task

Updating consists of adding new composition units to an application, replacing or removing composition units, or mapping composition units to different deployment targets.

You can add an asset or shared library composition unit to multiple business-level applications. However, each composition unit for the same asset must have a unique composition unit name. You can add a business-level application composition unit to more than one business-level application.

This topic describes how to update business-level applications using the administrative console. Alternatively, you can use programming or the wsadmin tool.

- Delete composition units from your business-level application.
 1. Go to the business-level application settings page.

Click **Applications** → **Application Types** → **Business-level applications** → *application_name* in the console navigation tree.
 2. Select each composition unit of the application that you want to delete.
 3. Click **Delete**.
 4. On the Delete composition unit from business-level application panel, confirm the deletion and click **OK**.
- Add new or updated assets, shared libraries, or other business-level applications to your business-level application.
 1. Update asset binary files or shared libraries as needed.
 2. If you are adding new assets that are not registered with the product management domain, import the assets.
 3. If you are updating existing assets, use the **Update** option to update asset files.
 4. On the business-level application settings page, specify the type of composition unit to add.
 - To add an asset, under **Deployed assets**, click **Add** → **Add Asset**.
 - To add a shared library, under **Deployed assets**, click **Add** → **Add Shared Library**.
 - To add a business-level application, under **Business-level applications**, click **Add**.
 5. On the New composition unit panel, select a unit from the list of available units, and then click **Continue**.
 6. On the Set options panel, change the composition unit settings as needed, and then click **Next**.
 7. On the Map composition unit to a target panel, change the deployment target as needed, and then click **Next**.

This panel is not shown when you add a business-level application.
 8. On the Summary page, click **Finish**.
 9. If the product adds the unit successfully, click **Manage application**.

If the unit addition is not successful, read the messages, and try adding the unit again. Correct the errors noted in any messages.
 10. On the Adding composition unit to the business-level application page, click **Save**.
 11. Repeat these steps to add any other assets, shared libraries, or applications needed by the business-level application.

The business-level application settings page displays the configuration unit names.

- Map composition units to different deployment targets.
 1. On the composition unit settings page, select the composition unit that you want to change.
 2. Under **Current targets**, click **Modify Target**.
 3. On the Map targets page, change the target.
 - a. From the list of available clusters and servers, select a different deployment target.
 - b. Click >> to add the deployment target to the **Selected** list.

- c. To remove a deployment target from the **Selected** list, select the target and click <<.
- d. Click **OK**.

The business-level application settings page displays the selected deployment target.

What to do next

Save the changes to your administrative configuration.

Deleting business-level applications

After an application no longer is needed, you can delete it.

About this task

Deleting a business-level application removes the application from the product configuration repository and it deletes the application binaries from the file system of all nodes where the application files are installed.

1. Go to the Business-level applications page.

Click **Applications** → **Application Types** → **Business-level applications** in the console navigation tree.

2. If you need to retain a copy of the application, back up composition units of the application.
3. Delete composition units of the application.
 - a. On the Business-level applications page, click the name of the business-level application that you want to delete.
 - b. On the business-level application settings page, select each composition unit of the application.
 - c. Click **Delete**.
 - d. On the Delete composition unit from Business-level application panel, confirm the deletion and click **OK**.

Deleting a configuration unit removes the configuration from the *profile_root/config/cells/cell_name/cus* directory.

4. Delete the business-level application.
 - a. Select the application that you want to delete.
 - b. Click **Delete**.

Unless the application is used by another business-level application, deleting a business-level application removes the configuration from the *profile_root/config/cells/cell_name/blas* directory.

5. On the Delete business-level application panel, confirm the deletion and click **OK**.
6. Save changes made to the administrative configuration.

Results

On single-server products, application binaries are deleted after you save the changes.

What to do next

If using the administrative console **Delete** options does not fully delete a business-level application or its configuration units, you can delete the business-level application and its configuration units manually from a stand-alone server. Suppose you want to delete a business-level application named ExampleBLA, and ExampleBLA is not used by another business-level application. Complete the following steps to manually delete the ExampleBLA configurations from the blas and cus directories:

1. Delete the *profile_root/config/cells/cell_name/blas/ExampleBLA* directory.
2. Delete the *profile_root/config/cells/cell_name/cus/ExampleBLA* directory.

3. Save changes made to the administrative configuration.

Chapter 10. Administering business-level applications using programming

You can use the command framework programming to create, edit, update, start, stop, delete, export, import, and query information about business-level applications. A business-level application defines an enterprise-level application.

Before you begin

This task assumes a basic familiarity with the command framework. Read about the command framework in the application programming interfaces documentation.

About this task

Besides creating, editing, updating, starting, stopping, deleting, exporting, importing, and querying information about business-level applications using programming, you can do these tasks using the administrative console or the wsadmin scripting tool.

1. Perform any of the following tasks to administer your business-level applications using programming.
 - a. Create an empty business-level application.

You typically create an empty business-level application and then add assets or business-level applications as composition units to the empty business-level application.
 - b. Import an asset.

You can import an asset to register the asset with the product and optionally store the asset in the product repository so that you can later use the asset in a business-level application. An asset represents at least one binary file that implements business logic.
 - c. Add a composition unit.

You can add an asset to a business-level application by creating a composition unit for the asset. A composition unit is typically created from an asset and contains configuration information that makes the asset runnable.
 - d. Start a business-level application.

You can start a business-level application, which starts each composition unit in that business-level application. Each composition unit is started on the respective targets on which the business-level application is deployed.
 - e. Stop a business-level application.

You can stop a business-level application, which stops each composition unit in that business-level application. Each composition unit is stopped on the respective targets on which the business-level application is deployed.
 - f. Check the status of a business-level application.

You can check the status of an entire business-level application. You can also limit the status to a particular composition unit of a business-level application, a specific deployment target, or check the status of the composition unit and the deployment target at the same time.
 - g. Delete a business-level application.

You can delete a business-level application using programming. You might delete a business-level application if the application is not functioning correctly, no longer needed, and so on.
 - h. Delete an asset.

You can delete an asset from a business-level application using programming if the asset is not functioning correctly, the asset is no longer needed, and so on. An asset represents at least one binary file that implements business logic.
 - i. Delete a composition unit.

You can delete a composition unit from a business-level application if the composition unit is not functioning correctly, the composition unit is no longer needed, and so on. A composition unit is typically created from a business-level application or an asset and contains configuration information that makes the asset runnable.

j. Export an asset.

You can export an asset from the current session so that you can back up the asset, import the asset to another session, and so on. An asset represents at least one binary file that implements business logic.

k. List assets.

You can list the assets that have been imported to the current workspace so that you can do further asset administration, such as deleting or exporting assets. An asset represents at least one binary file that implements business logic.

l. List composition units.

You can list the composition units for a specific business-level application in a session so that you can do further composition unit administration, such as deleting or adding composition units. A composition unit is typically created from a business-level application or an asset and contains configuration information that makes the asset runnable.

m. List business-level applications.

You can list the business-level applications of a session so that you can do further business-level application administration such as deleting a business-level application. A business-level application is an administrative model that captures the definition of an enterprise-level application so that you can perform specific business functions, such as accounting.

n. Edit a composition unit.

You can edit the configuration information in a composition unit of a business-level application if, for example, you want to change which modules in the composition unit are configured to run in which targets. A composition unit is typically created from a business-level application or an asset and contains configuration information that makes the asset runnable.

o. Edit an asset.

You can edit the information of an asset, for example, its destination location, its relationship with other assets, and so on. An asset represents at least one binary file that implements business logic.

p. Edit a business-level application.

You can edit the information of a business-level application such as its description. A business-level application is an administrative model that captures the entire definition of an enterprise-level application.

q. Update an asset.

You can update an asset by adding, deleting, or updating a single file or Java Platform, Enterprise Edition (Java EE) module, or by merging multiple files or Java EE modules into an asset. You can also update an asset by replacing the entire asset.

r. View a composition unit.

You can view the composition unit information so that you can do other tasks associated with the composition unit, such as editing an asset or deleting a composition unit. A composition unit is typically created from a business-level application or an asset and contains configuration information that makes the asset runnable.

s. View an asset.

You can view the asset information so that you can do other tasks associated with the asset, such as editing or exporting an asset. An asset represents at least one binary file that implements business logic.

t. View a business-level application.

You can view business-level application information such as the description so that you can do other tasks associated with the business-level application, such as editing the business-level application. A business-level application is an administrative model that captures the entire definition of an enterprise-level application.

u. List control operations.

You can list the control operations of a business-level application or a composition unit for a session. You use control operations, such as start or stop, to change or query the runtime environment of a business-level application or a composition unit.

2. Save your changes to the master configuration repository.
3. Synchronize changes to the master configuration across the nodes for the changes to take effect.

Results

Depending on which tasks you complete, you have created, edited, updated, started, stopped, deleted, exported, imported, or queried information about business-level applications.

What to do next

If you have further business-level application updates, you can do the updates through programming, the administrative console, or the wsadmin scripting tool.

Creating an empty business-level application using programming

You can create an empty business-level application, and then add assets or business-level applications as composition units to the empty business-level application.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

You can create an empty business-level application using programming, the administrative console, or the wsadmin tool.

About this task

Perform the following steps to create an empty business-level application using programming. In your code that creates the empty business-level application, you must provide the name parameter. The name parameter specifies the name of the business-level application that you create.

1. Connect to the application server.

The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.

The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.

Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is

passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Use the command manager that you created in a previous step to create and set up the command that creates an empty business-level application.

The command name is `createEmptyBLA`. The name parameter is a required parameter that you use to specify the name of the business-level application. You can optionally provide the description parameter to provide a description of the newly created business-level application.

6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the `execute` method in the asynchronous command client to run the command that creates an empty business-level application.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, the empty business-level application is created.

Example

The following example shows how to create an empty business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.exception.AdminException;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class CreateEmptyBLA {

    public static void main(String[] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.

            String host = "localhost";
            String port = "8880";
            // Change to your port number if it is
            // not 8880.
```



```

Properties config = new Properties();
config.put(AdminClient.CONNECTOR_HOST, host);
config.put(AdminClient.CONNECTOR_PORT, port);
config.put(AdminClient.CONNECTOR_TYPE,
    AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println("Config: " + config);
AdminClient soapClient =
    AdminClientFactory.createAdminClient(config);

// Create the command manager.
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager:
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler
// for listening to command notifications.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the listener with
// null for the AsyncCommandClient object that follows.

AsyncCommandClient asyncCmdClientHelper = new
    AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that creates an empty
// business-level application.
String cmdName = "createEmptyBLA";

AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Create an empty
    // business-level application using
    // the session created.
System.out.println("\nCreated " + cmdName);

// Set the name command parameter.
String blaName = "bla1";
cmd.setParameter("name", blaName);

System.out.println("\nSet name parameter to "
    + cmd.getParameter("name"));

// Uncomment the following lines to set the description of
// the business-level application being created:
//
// String blaDescription = "description for bla1";
// cmd.setParameter("description", blaDescription);
// System.out.println("\nSet description parameter to " +
//     cmd.getParameter("description"));

// Call the asynchronous command client to

```

```

// process the command parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Call the asynchronous command client to run the command.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted command execution");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
            + "with result\n" + result.getResult());
    } else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification.
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

You can add business-level applications or assets as composition units into the newly created business-level application. Alternatively, you can add the newly created business-level application to other business-level applications.

Importing an asset using programming

You can import an asset to register the asset with the product and optionally store the asset in the product repository so that you can later use that asset in a business-level application. An asset represents at least one binary file that implements business logic.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

You can import an asset using programming, the administrative console, or the wsadmin tool.

About this task

When you import an asset, you register the asset with the product and optionally store the asset in the product repository.

You must provide a file path to the source that you are importing. Specify an absolute path name to the source, as the behavior for a relative path is unpredictable.

You can specify a destination location from where the application server reads the asset file while starting a composition unit created from the asset. The asset is copied to this location when the configuration session is saved after the asset is imported. The default asset destination is *profile_root/installedAssets/asset_name*.

You can optionally specify a storage type of FULL, METADATA, or NONE. The default value is FULL, which means that the asset and associated meta data are stored in the product asset repository. If you specify a storage type of METADATA, the asset is not copied to the product repository, but associated meta data is stored in the product repository. If you specify a storage type of NONE, neither the asset nor the asset meta data is stored in the product asset repository. For storage types of METADATA and NONE, the asset is expected to reside at the destination file path. Storage types of METADATA and NONE are typically used by development tools which enable iterative development on the copy of the asset in the directory structure of the tool.

Perform the following steps to import an asset using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager that you created in a previous step to set up the command that imports an asset.
The command name is `importAsset`. The source parameter is a required parameter that you use to specify the path to the asset. You can optionally provide the `storageType` parameter to specify how to save the asset in the configuration repository.
6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.
The command framework asynchronous command model requires this call.

7. Set up the command step parameters.

You can set parameters in the AssetOptions step that contains data about the asset such as its description, file permission, and relationship with other assets.

8. Call the asynchronous command client to run the command that imports an asset.

You might have created an asynchronous command handler to implement the AsyncCommandHandlerIF interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

9. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the command.getCommandResult method.

Results

After you successfully run the code, the asset is imported.

Example

The following example shows how to import an asset based on the previous steps.

Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;
import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;

import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.UploadFile;
import com.ibm.websphere.management.exception.AdminException;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class ImportAsset {

    public static void main (String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.

            String host = "localhost";
            String port = "8880"; //Change to your port number if it is not
            //8880.

            Properties config = new Properties();
            config.put (AdminClient.CONNECTOR_HOST, host);
            config.put (AdminClient.CONNECTOR_PORT, port);
            config.put (AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println ("Config: " + config);
```

```

AdminClient soapClient =
    AdminClientFactory.createAdminClient(config);

// Create the command manager.
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager:
//
//CommandMgr cmdMgr = CommandMgr.getCommandMgr();

System.out.println("\nCreated command manager");

// Optionally create the asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Setup the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the following listener with
// null for the AsyncCommandClient object.

AsyncCommandClient asyncCmdClientHelper = new
    AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

String cmdName = "importAsset";
UploadFile assetSource = new
UploadFile("/sources/test5.zip"); //Change to the directory of your sources.

// Create the command to import an asset.
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); //import the asset using
//the session created
System.out.println("\nCreated " + cmdName);

// Set the source command parameter.
cmd.setParameter("source", assetSource);
System.out.println("\nSet source parameter to " +
    cmd.getParameter("source"));

// Uncomment the following line to set the storage type to
// a value of STORAGETYPE_META or STORAGETYPE_NONE instead of
// the default of STORAGETYPE_FULL:
//
//cmd.setParameter("storageType,
//    CommandConstants.STORAGETYPE_NONE);

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
}
catch(Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
}

```

```

        System.exit(-1);
    }

    // Set up the step parameters for the AssetOptions step.
    String stepName = "AssetOptions";
    CommandStep step = ((TaskCommand) cmd).gotoStep(stepName);

    // The new asset name must contain the
    // same extension as the original .zip file name.
    String assetNewName = "asset1.zip";

    // If you override the default destination, include the
    // entire path with the file name for the new destination.
    String destName = "/websphere/asset/installDir/asset1.zip";
    for (int i = 0; i < step.getNumberOfRows(); i++) {
        // The following lines change the name and destination
        // step parameters. Other step parameters that you can
        // use follow, but are commented out.
        // Change your set
        // of step parameters as required by your scenario.

        // Set the name.
        step.setParameter("name", assetNewName, i);
        System.out.println("\nSet name parameter to " +
            step.getParameter("name", i));

        // Set the destination.
        step.setParameter("destination", destName, i);
        System.out.println("\nSet destination parameter to " +
            step.getParameter("destination", i));

        // Set the description.
        //String desc = "description for asset1.zip";
        //step.setParameter("description", desc, i);
        //System.out.println("\nSet description parameter to " +
        //    step.getParameter("description", i));

        // Set the validation.
        //String validate = "Yes";
        //step.setParameter("validate", validate, i);
        //System.out.println("\nSet validate parameter to " +
        //    step.getParameter("validate", i));

        // Set the file permission.
        //String filePermission = ".*\\.dll=755";
        //step.setParameter("filePermission", filePermission, i);
        //System.out.println("\nSet filePermission parameter to " +
        //    step.getParameter("filePermission", i));

        // Set the type aspect parameter value.
        // Format for a typeAspect: WebSphere:spec=xxx,version=n.n+
        // Websphere:spec=xxx,version=n.n.
        //String typeAspect = "";
        //step.setParameter("typeAspect", typeAspect, i);
        //System.out.println("\nGet typeAspect: " +
        //    step.getParameter("typeAspect", i));

        // Set the relationship parameter.
        // The relationship parameter declares dependency
        // relationships on other assets. The parameter value
        // is a list which contains the ID of each asset declared
        // as a dependency. Each ID in the list is separated by
        // a "plus" sign ("+").
        //
        // Only assets which are Java archives can be referenced in
        // dependency relationships. An asset is a Java archive if

```

```

        // it has a type aspect identifying it as such.
        //
        // If an asset declared as a dependency does not exist or
        // does not have a Java archive type aspect, it is ignored
        // and no dependency on the asset is registered in the
        // asset's configuration.
        //
        //String relationship =
        //    "assetname=shared.zip+assetname=shared2.zip";
        //step.setParameter("relationship", relationship, i);
        //System.out.println("\nGet relationship: " +
        //    step.getParameter("relationship", i));
    }

    // Call the asynchronous command client that imports the asset.
    asyncCmdClientHelper.execute(cmd);
    System.out.println("\nCompleted running of command");

    // Check the command result.
    CommandResult result = cmd.getCommandResult();
    if (result != null) {
        if (result.isSuccessful()) {
            System.out.println("\nCommand ran successfully " +
                "with result\n" + result.getResult());
        } else {
            System.out.println("\nCommand ran with " +
                "exception");
            result.getException().printStackTrace();
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}
}

```

What to do next

Add a composition unit to a business-level application using the asset that you imported. An asset included in a business-level application is represented by a composition unit.

Adding a composition unit using programming

You can add an asset to a business-level application by creating a composition unit for the asset. A composition unit is typically created from a business-level application or an asset and contains configuration information that makes the asset runnable.

Before you begin

Before you can add a composition unit to a business-level application, you must have created an empty business-level application and imported an asset.

You can add a composition unit to a business-level application using programming, the administrative console, or the wsadmin tool.

About this task

When you add a composition to a business-level application, the composition unit is configured for the specified business-level application. The composition unit cannot be shared with other business-level applications.

Perform the following steps to add a composition unit to a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager that you created in a previous step to create and set up the command that adds a composition unit.
The command name is `addCompUnit`. The `blID` and `cuSourceID` parameters are required parameters that you use to specify composition unit source to be added to the business-level application. Examples of composition unit source are an asset or a business-level application. You can optionally provide deployable units for the composition unit through the `deplUnit` parameter. If the `cuSourceID` parameter is a Java Platform, Enterprise Edition (Java EE) asset, you can optionally use the `cuConfigStrategyFile` parameter or the `defaultBindingOptions` parameter to specify the default bindings. The `defaultBindingOptions` parameter must match the binding options available for this Java EE asset. To view a list of binding options available for this Java EE asset, look at the `AssetOptions` step in the `viewAsset` command. Specify each binding option in an `option_name=option_value` pair, with multiple pairs separated by a `#` character.
6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.
The command framework asynchronous command model requires this call.
7. Set up the command step parameters.
You can set up composition unit information through various steps. The `CUOptions` step contains data about the composition unit such as its description, starting weight, and start and restart behavior. The `MapTargets` step contains target information about where the composition unit is to be deployed. The `RelationshipOptions` step contains shared library composition units on which this composition unit has dependencies. The `ActivationPlanOptions` step allows you to specify runtime components for each

deployable unit. The `CreateAuxCUOptions` step contains assets on which this composition unit has dependencies. You can set up parameters in these steps.

8. Call the asynchronous command client to run the command that adds a composition unit to a business-level application.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

9. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, the composition unit is added to the business-level application.

Example

The following example shows how to import an asset based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class AddCompUnit {

    public static void main(String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local command
            // manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the
            // local command manager.

            String host = "localhost";
            String port = "8880"; // Change to your port number if it is
            // not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create the command manager
```

```

CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager:
//
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();

System.out.println("\nCreated command manager");

// Optionally create the asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the following listener with
// null for the AsyncCommandClient object.
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command to add a composition unit to a business-level application.
String cmdName = "addCompUnit";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Add the composition unit using
// the session created.
System.out.println("\nCreated " + cmdName);

// Set the blaID command parameter.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter accepts an
// incomplete ID as long as the incomplete
// ID can resolve to a unique business-level application.
String blaID = "bla1";
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
+ cmd.getParameter("blaID"));

// Set the cuSourceID command parameter.
// Examples of valid formats for the cuSourceID parameter:
// If the source is an asset, examples are:
// - aName
// - assetname=aName
// - WebSphere:assetname=aName
// If the source is another business-level application,
// examples are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// The cuSourceID command parameter
// accepts an incomplete ID as long as the incomplete
// ID can resolve to a unique asset or business-level application.
String cuSourceID = "assetname=asset1.zip";
cmd.setParameter("cuSourceID", cuSourceID);

```

```

System.out.println("\nSet cuSourceID parameter to "
    + cmd.getParameter("cuSourceID"));

// Set the deplUnits command parameter.
// If the deployable units of an asset are, for example, a.jar and
// b.jar, then when you run the addCompUnit command you can
// specify deplUnits as a.jar+b.jar. You can specify the whole
// list, a subset of that list, or "default" to create this composition
// unit as a shared library. If the deplUnits parameter is not specified,
// the deployable units are set the same as that of their asset.
String deplUnits = "default";
cmd.setParameter("deplUnits", deplUnits);

System.out.println("\nSet deplUnits parameter to "
    + cmd.getParameter("deplUnits"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Set up the step parameters for the CUOptions step.
// The CUOptions step contains the following arguments:
// description - description for the composition unit
// startingWeight - starting weight for the composition
// unit within the business-level application. The default is 1.
// startedOnDistributed - to start composition unit upon distribution
// to target nodes. The default is false.
// restartBehaviorOnUpdate - restart behavior for a composition unit when
// updating the composition unit.
// The default is DEFAULT. Valid values are DEFAULT, ALL, and NONE.
String stepName = "CUOptions";
CommandStep step = ((TaskCommand) cmd).gotoStep(stepName);

// Composition unit name:
String name = "cu1";

// Composition unit description:
String description = "cu1 description";

for(int i = 0; i < step.getNumberOfRows(); i++) {
    // The following lines change the composition unit name and
    // description step parameters of the CUOptions step. Change
    // your set of step parameters as required for your
    // scenario.

    // Set the name.
    step.setParameter("name", name, i);
    System.out.println("\nSet name parameter to " +
        step.getParameter("name", i));

    // Set the description.
    step.setParameter("description", description, i);
    System.out.println("\nSet description parameter to " +
        step.getParameter("description", i));
}

// Set up the step parameters for the MapTargets step.
stepName = "MapTargets";
step = ((TaskCommand) cmd).gotoStep(stepName);

```

```

// Specify the targets to deploy the composition unit.
// The default is server1. Use the + character to
// specify multiple targets.
String server = "server1";

for(int i = 0; i < step.getNumberOfRows(); i++) {
    // The following lines change the composition unit and
    // server step parameters of the
    // MapTargets step. Change your set of step parameters
    // as required for your scenario.

    // Set the server.
    step.setParameter("server", server, i);
    System.out.println("\nSet server parameter to " +
        step.getParameter("server", i));
}

// The addCompUnit command might contain the
// CreateAuxCUOptions, RelationshipOptions and ActivationPlanOptions
// steps, depending on the asset content of the assets imported.
// The CreateAuxCUOptions step is available if the cuSourceID value
// is an asset. The asset includes an asset relationship to an
// asset that does not have a matching composition unit in the
// business-level application.
//
// If the CreateAuxCUOptions step is available, the selected
// deployable units of the source asset of the "primary" composition
// unit (that is, the composition unit being added) have dependencies
// on other assets for which there are no matching composition units
// in the business-level application. A "secondary" composition unit will be created for each
// of those asset dependencies.
//
// Each CreateAuxCUOptions row corresponds to one dependency
// relationship declaration. Each row consists of parameter values
// for the dependency relationship. Some parameters are read-only and
// some of them are editable. To edit parameter values, use the same
// approach as that used to edit parameter values in the CUOptions step.
//
// The parameters for this step include:
//
// deplUnit - The name of the deployable unit which has the
//             dependency. (Read-only.)
// inputAsset - The asset ID for the source asset of the primary
//              composition unit. (Read-only.)
// cuID - The name of the secondary composition unit to create.
// matchTarget - Specifies whether the server target for the secondary
//              composition unit is to match the server target for
//              the primary composition unit. The default value
//              is "true". If the value is set to "false", the
//              secondary composition unit will be created with no
//              target. The target on the secondary composition unit
//              can be set at a later time with the editCompUnit
//              command.
//
// If the RelationshipOptions step is available, the selected
// deployable units of the source asset of the "primary" composition
// unit (that is, the composition unit being added) have dependencies
// on other assets for which there are matching "secondary" composition
// units in the business-level application. The RelationshipOptions step is much like
// CreateAuxCUOptions except that the required secondary composition
// units already exist. Also, each RelationshipOptions row maps one
// deployable unit to one or more secondary composition units, whereas,
// each CreateAuxCUOptions row maps one deployable unit to one
// asset dependency.
//
// Each RelationshipOptions row corresponds to one deployable unit

```

```

// with one or more dependency relationships and consists of
// parameter values for the dependency relationships. Some parameters
// are read-only and some of them are editable. To edit parameter
// values, use the same approach as that used to edit parameter values
// in the CUOptions step.
//
// The parameters for this step include:
//
// deplUnit - The name of the deployable unit which has the
//            dependency. (Read-only.)
// relationship - Composition unit dependencies in the form of a
//               list of composition unit IDs. Composition unit
//               IDs are separated by a "plus" sign (+). Each ID
//               can be fully or partially formed as shown with the
//               following examples:
//               WebSphere:cuname=SharedLib1.jar
//               WebSphere:cuname=SharedLib.jar
//               SharedLib.jar
// matchTarget - Specifies whether the server target for the secondary
//               composition units are to match the server target for
//               the primary composition unit. The default value
//               is "true". If the value is set to "false", the
//               secondary composition unit will be created with no
//               target. The target on the secondary composition unit
//               can be set at a later time with the editCompUnit
//               command.
// The addCompUnit command contains the ActivationPlanOptions step.
// The user can set the ActivationPlanOptions step parameters
// similar to the step parameters for the CUOptions step in
// the previous examples. The arguments for this step include:
// deplUnit - deployable unit URI (read only parameter)
// activationPlan - specifies a list of runtime components in the
//                 format of specname=xxxx
//
// Run the command to add the composition unit.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
    }
}

```

```
        System.out.println("\nEXAMPLE: notification received: " +
                           notification);
    }
}
```

What to do next

Start the business-level application to which you added the composition unit. Complete administrative tasks such as viewing or deleting the composition unit.

Starting a business-level application using programming

You can start a business-level application, which starts each composition unit in that business-level application. Each composition unit is started on the respective targets on which the business-level application is deployed.

Before you begin

Before you can start a business-level application, you must have created an empty business-level application, imported an asset, and added a composition unit to the business-level application.

You can start a business-level application using programming, the administrative console, or the wsadmin tool.

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

About this task

You must specify the blaID parameter of the business-level application that you start.

Perform the following steps to start a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager that you created in a previous step to set up the command that starts a business-level application.
The command name is startBLA. The blaID parameter is a required parameter to specify the business-level application to start.
6. Call the processCommandParameters method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to run the command that starts a business-level application. You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.
8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, the business-level application is started.

Example

The following example shows how to start a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class startBLA {

    public static void main(String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local command
            // manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if
            // you use the local command manager.

            String host = "localhost";
            String port = "8880"; // Change to your port number if it is
                                // not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create the command manager
            CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);
```

```

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager:
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler
// for listening to command notifications.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the listener with
// null for the AsyncCommandClient object that follows.
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that starts the business-level application.
String cmdName = "startBLA";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Start a business-level
//application using the session created.
System.out.println("\nCreated " + cmdName);

// (required) Set the blaID parameter.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter
// accepts an incomplete ID as long as the incomplete
// ID can resolve to a unique business-level application.
String blaID = "bla1";
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
+ cmd.getParameter("blaID"));

// Call asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Call the asynchronous command client to run the command command.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running the command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {

```



```

        if (result.isSuccessful()) {
            System.out.println("\nCommand ran successfully "
                + "with result\n" + result.getResult());
        }
        else {
            System.out.println("\nCommand ran with " +
                "Exception");
            result.getException().printStackTrace();
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}
}

```

What to do next

Your users can access the business-level application that you started.

Stopping a business-level application using programming

You can stop a business-level application, which stops each composition unit in that business-level application. Each composition unit is stopped on the respective targets on which the business-level application is deployed.

Before you begin

Before you can stop a business-level application, you must have created an empty business-level application, imported an asset, added a composition unit to the business-level application, and started the business-level application.

About this task

You can stop a business-level application using programming, the administrative console, or the wsadmin tool. This topic describes how to stop a business-level application using programming.

Perform the following steps to stop a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.

Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.

4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Use the command manager that you created in a previous step Create and set up the command that stops a business-level application.

The command name is stopBLA. The blaID parameter is a required parameter to specify the business-level application to stop.

6. Call the processCommandParameters method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to run the command that stops a business-level application.

You might have created an asynchronous command handler to implement the AsyncCommandHandlerIF interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the command.getCommandResult method.

Results

After you successfully run the code, the business-level application is stopped.

Example

The following example shows how to stop a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class stopBLA {

    public static void main(String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local command
            // manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
```

```

// to get the soapClient soap client if
// you use the local command manager.

String host = "localhost";
String port = "8880"; // Change to your port number if it is
                    // not 8880.

Properties config = new Properties();
config.put(AdminClient.CONNECTOR_HOST, host);
config.put(AdminClient.CONNECTOR_PORT, port);
config.put(AdminClient.CONNECTOR_TYPE,
          AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println("Config: " + config);
AdminClient soapClient =
    AdminClientFactory.createAdminClient(config);

// Create the command manager
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager.
//
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler
// for listening to command notifications.
// Comment out the following line if no further handling
// of command notification is required.
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the listener with
// null for the AsyncCommandClient object that follows.
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that stops the business-level application.
String cmdName = "stopBLA";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Stop a business-level
    //application that is using the session created.
System.out.println("\nCreated " + cmdName);

// (required) Set the blaID parameter.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter
// accepts an incomplete ID as long as the incomplete
// ID can resolve to a unique business-level application.
String blaID = "bla1";
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
    + cmd.getParameter("blaID"));

```

```

// Call asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Call the asynchronous command client to run the command command.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}
}
}

```

```

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

Complete administrative tasks on the business-level application, such as editing an asset or a composition unit that is contained in the business-level application.

Checking the status of a business-level application using programming

You can check the status of an entire business-level application. You can also limit the status to a particular composition unit of a business-level application, a specific deployment target, or check the status of the composition unit and the deployment target at the same time.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interface documentation.

Before you can check the status of a business-level application or a composition unit, you must have created the business-level application.

You can check the status of a business-level application using programming, the administrative console, or the wsadmin tool.

About this task

You must provide the blaID parameter to specify the business-level application that you are viewing.

Perform the following tasks to view a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Create and set up the getBLAStatus command to check the status of a business-level application.
 - a. Set the blaID parameter for the business-level application whose status you want to check.
 - b. Optionally set the cuID parameter if you want to narrow the scope of the query to a single composition unit.
 - c. Optionally set the targetID if you want to narrow the scope of the query to a single target server process or cluster.
6. Call the processCommandParameters method in the asynchronous command client to process the command parameters.
The command framework asynchronous command model requires this call.
7. Call the asynchronous command client to run the command to check the status of the business-level application.
You could have created an asynchronous command handler to implement the AsyncCommandHandlerIF interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.
8. Check the command result when the command completes.
When the command finishes running, control is returned to the caller. You can then check the result by calling the command.getCommandResult method.

Results

After you successfully run the code, you can check the status of an entire business-level application, if you chose not to limit the status. If you chose options to limit the status, you could check the status to a particular composition unit of a business-level application, a specific deployment target, or check the status of the composition unit and the deployment target at the same time.

The smallest unit of status data that the system maintains is for a single composition unit in a single server or cluster member process. Business-level application status can be based on one or more composition units, each having one or more targets, with targets potentially consisting of clusters with multiple member processes. Therefore, the single status value returned from the `getBLAStatus` command is a compilation of individual status data for all composition units on all target process within the scope of the status query. The following table describes how individual status data is compiled into a single status value. The term *composition unit instance* used in the table refers to a composition unit on a single server or single cluster member process.

Status	Description
<code>ExecutionState.STARTED</code>	All composition unit instances within the scope of the query have been started.
<code>ExecutionState.STOPPED</code>	All composition unit instances within the scope of the query have not been started or have been stopped.
<code>ExecutionState.PARTIAL_START</code>	Some composition unit instances within the scope of the query have a status of <code>ExecutionState.STARTED</code> and some have a status of <code>ExecutionState.STOPPED</code> .
<code>ExecutionState.UNKNOWN</code>	Status data for at least one composition instance within the scope of the query cannot be obtained for some reason.

Example

The following example shows how to check the status of a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class EditBLA {

    public static void main(String[] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local command manager.
            // Comment out the following lines to get the soapClient soap client if
            // you are going to use the local command manager. You would
            // comment out the lines to and including
            // CommandMgr cmdMgr =
            // CommandMgr.getClientCommandMgr(soapClient);
```

```

String host = "localhost"; // Change to your host if it is not localhost.
String port = "8880"; // Change to your port number if it is not 8880.

Properties config = new Properties();
config.put(AdminClient.CONNECTOR_HOST, host);
config.put(AdminClient.CONNECTOR_PORT, port);
config.put(AdminClient.CONNECTOR_TYPE,
    AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println("Config: " + config);
AdminClient soapClient =
    AdminClientFactory.createAdminClient(config);

// Create command manager.
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager.
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();

System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required.
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
// This example creates a new session. You can replace the
// following code to use an existing session that has been
// created.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the listener with
// null for the following AsyncCommandClient object.
AsyncCommandClient asyncCmdClientHelper = new
    AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command.
String cmdName = "getBLAStatus";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Check the status
                                // using the session
                                // created
System.out.println("\nCreated " + cmdName);

// Set the required blaID parameter
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
String blaID = "MyBLA"; // Replace the MyBLA value with your
                        // blaID value.
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
    + cmd.getParameter("blaID"));

// Optionally set the cuID parameter.

```

```

String cuID = "myCU.zip"; // Replace the myCU.zip value with your
                        // cuID value.
cmd.setParameter("cuID", cuID);

System.out.println("\nSet cuID parameter to "
    + cmd.getParameter("cuID"));

// Optionally set the targetID parameter.
// The format of the targetID parameter for a cluster
// is WebSphere:cluster=cluster1
String targetID = "WebSphere:node=node1,server=server1"; // Replace
// this with your targetID value.
cmd.setParameter("targetID", targetID);

System.out.println("\nSet targetID parameter to "
    + cmd.getParameter("targetID"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted processCommandParameters");
} catch (Throwable th) {
    System.out.println("Throwing an exception from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Run the command to check the status of the
// business-level application.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted command execution");

CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand executed successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand executed with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification.
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```


What to do next

You can use the results of the status check to perform other tasks. For instance, if the results indicate that none of the composition units is started, you could start the business-level application.

Deleting a business-level application using programming

You can delete a business-level application using programming. You might delete a business-level application if it is not functioning correctly, it is no longer needed, and so on.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

Before you can delete a business-level application, you must have created an empty business-level application. You can optionally have added assets or business-level applications as composition units to the empty business-level application. All the composition units in the business-level application must be deleted using the `deleteCompUnit` command before you delete the business-level application. Other business-level applications cannot reference the business-level application that you are deleting. Otherwise, the deletion fails.

You can delete a business-level application using programming, the administrative console, or the `wsadmin` tool.

About this task

You must specify the `blaID` parameter of the business-level application that you are deleting. You might delete a business-level application if it is not functioning correctly, no longer needed, and so on.

Perform the following steps to delete a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager you created in a previous step to create and set up the command that deletes the business-level application.
The command name is `DeleteBLA`. The `blaID` parameter is a required parameter to specify the business-level application to delete.
6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.
The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to run the command that deletes the business-level application. You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.
8. Check the command result when the command completes. When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, the business-level application is deleted.

Example

The following example shows how to delete a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class DeleteBLA {

    public static void main(String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.

            String host = "localhost";
            String port = "8880"; // Change to your port number if it is
            // not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create the command manager.
            CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

            // Comment out the previous lines to create a client command
```

```

// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager:
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Setup the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace listener with
// null for the following AsyncCommandClient object:
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that deletes the business-level application.
String cmdName = "deleteBLA";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Delete the business-level
// application using the session created.
System.out.println("\nCreated " + cmdName);

// Set the blaID parameter to the business-level application to delete.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter accepts an incomplete ID as long as the incomplete
// ID can resolve to a unique business-level application.
String blaID = "bla1";
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
+ cmd.getParameter("blaID"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Call the asynchronous command client to run the command.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of the command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "

```

```

        + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

```

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

You can complete other tasks associated with business-level applications, such as creating other business-level applications, stopping and starting business-level applications, and so on.

Deleting an asset using programming

You can delete an asset from a business-level application using programming if the asset is not functioning correctly, the asset is no longer needed, and so on. An asset represents at least one binary file that implements business logic.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interface documentation.

Before you can delete an asset, you must have imported the asset. All the composition units associated with the asset must be deleted using the `deleteCompUnit` command before you delete the asset. Otherwise, you have to force the deletion. If you do not force the deletion, the deletion fails. If any other composition units have a dependency on a composition unit being deleted with the `force` option, the deletion fails. After all dependencies on the composition unit are removed, the `force` option succeeds.

About this task

You can delete an asset using programming, the administrative console, or the `wsadmin` tool. Use this topic to delete an asset using programming.

You must specify the `assetID` parameter of the asset that you are deleting. You might delete an asset if it is not functioning correctly, it is no longer needed, and so on.

Perform the following tasks to delete an asset using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager that you created in a previous step to create and set up the command that deletes an asset.
The command name is `deleteAsset`. The `assetID` parameter is a required parameter to specify the asset to delete. You can optionally specify the `delete` parameter to force deletion of an asset if composition units are still associated with the asset.
6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.
The command framework asynchronous command model requires this call.
7. Call the asynchronous command client to run the command that deletes an asset.
You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.
8. Check the command result when the command completes.
When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, the asset is deleted.

Example

The following example shows how to delete an asset from a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;
```

```

public class DeleteAsset {

    public static void main(String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.

            String host = "localhost";
            String port = "8880"; // Change to your port number if it is
                // not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create the command manager.
            CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

            // Comment out the previous lines to create a client command
            // manager if you are using a local command manager.
            // Uncomment the following line to create a local command
            // manager.
            //
            // CommandMgr cmdMgr = CommandMgr.getCommandMgr();
            System.out.println("\nCreated command manager");

            // Optionally create an asynchronous command handler.
            // Comment out the following line if no further handling
            // of command notification is required.
            AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

            // Create an asynchronous command client.

            // Set up the session.
            String id = Long.toHexString(System.currentTimeMillis());
            String user = "content" + id;
            Session session = new Session(user, true);

            // If no command handler is used, replace listener with
            // null for the following AsyncCommandClient object:
            AsyncCommandClient asyncCmdClientHelper = new
            AsyncCommandClient(session, listener);
            System.out.println("\nCreated async command client");

            // Create the command that deletes the asset.
            String cmdName = "deleteAsset";
            AdminCommand cmd = cmdMgr.createCommand(cmdName);
            cmd.setConfigSession(session); // Delete the asset from the
                // business-level application using the session created.
            System.out.println("\nCreated " + cmdName);

            // Set the assetID parameter to the asset that is to be
            // deleted.
            // Examples of valid formats for the assetID parameter are:

```

```

// - aName
// - assetname=aName
// - WebSphere:assetname=aName
// This parameter will accept an incomplete ID as long as
// the incomplete ID can resolve to a unique asset
// in the business-level application.
String assetID = "as1";
cmd.setParameter("assetID ", assetID );

System.out.println("\nSet assetID parameter to "
    + cmd.getParameter("assetID "));
// Uncomment the following line of code to set the force parameter
// to force the deletion even if there are composition units
// associated with this asset.
//
// cmd.setParameter("force", "true");

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Call the asynchronous command client to run the command.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running the command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

You can complete other steps associated with assets in business-level applications, such as adding or deleting other assets, listing assets, exporting assets, and so on.

Deleting a composition unit using programming

You can delete a composition unit from a business-level application if the composition unit is not functioning correctly, the composition unit is no longer needed, and so on. A composition unit is typically created from a business-level application or an asset and contains configuration information that makes the asset runnable.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

Before you can delete a composition unit, you must have created an empty business-level application, imported an asset, and added a composition unit to the business-level application. If other composition units depend on the composition unit that you are deleting and you do not use the force option, the deletion fails.

About this task

You can delete a composition unit using programming, the administrative console, or the wsadmin tool. This topic describes how to delete a composition unit using programming.

You must provide the blaID and culD parameters to specify the composition unit that you are deleting from the business-level application.

Perform the following tasks to delete a composition unit using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager that you created in a previous step to create and set up the command that deletes a composition unit.
The command name is deleteCompUnit. The blaID and culD parameters are required parameters. The culD parameter is used to specify the composition unit to delete from the business-level application, which is specified with the blaID. You can optionally provide the force parameter to force the deletion if other composition units depend on this composition unit.

6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to run the command that deletes a composition unit.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, the composition unit is deleted.

Example

The following example shows how to delete a composition unit from a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class DeleteCompUnit {

    public static void main(String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);
            // to get the soapClient soap client if
            // you use the local command manager.

            String host = "localhost";
            String port = "8880"; // Change to your port number if it is
                                // not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);
```

```

// Create the command manager.
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager:
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace listener with
// null for the following AsyncCommandClient object:
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that deletes the composition unit.
String cmdName = "deleteCompUnit";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Delete the composition unit from
// the business-level application
// using the session created.
System.out.println("\nCreated " + cmdName);

// Set the blaID parameter to the business-level application with
// the composition unit to delete.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter accepts an incomplete ID as long as
// the incomplete ID can resolve to a unique
// business-level application.
String blaID = "bla1";
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
+ cmd.getParameter("blaID"));

// Set the cuID parameter to the composition unit that is to be
// deleted.
// Examples of valid formats for the cuID parameter are:
// - name
// - cuname=name
// - WebSphere:cuname=name
// This parameter accepts an incomplete ID as long as the
// incomplete ID can resolve to a unique composition unit
// within the business-level application.
String cuID = "cu1";
cmd.setParameter("cuID", cuID);

System.out.println("\nSet cuID parameter to "
+ cmd.getParameter("cuID"));

```

```

// Uncomment the following line of code to set the force parameter
// to force the deletion even if other composition units depend
// on this composition unit.
//
// cmd.setParameter("force", "true");

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Call the asynchronous command client to run the command.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of the command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}
}
}

```

```

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

You can complete other tasks associated with the business-level application, such as adding or deleting other composition units, listing composition units, and so on.

Exporting an asset using programming

You can export an asset from the current session so that you can back up the asset, import the asset to another session, and so on. An asset represents at least one binary file that implements business logic.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

This task assumes that you have already imported an asset.

About this task

You can export an asset using programming, the administrative console, or the wsadmin tool. This topic describes how to export an asset using programming.

You must provide an `assetID` parameter value and a file name parameter value to export an asset. The `assetID` parameter identifies the asset you want to export. An asset ID can take a number of forms. The list below shows various forms for an asset named `asset1.jar`.

- `asset1.jar`
- `assetname=asset1.jar`
- `WebSphere:assetname=asset1.jar`

The `filename` parameter specifies a file system file name and location for the exported asset. Specify a fully qualified file path for the file name parameter because the results with relative path names are unpredictable. If you specify a file name parameter of a file that already exists, the file is overwritten with the exported asset.

Perform the following tasks to export an asset from a business-level application using programming.

1. Connect to the application server.

The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.

2. Create the command manager.

The command manager provides the functionality to create a new administrative command or query existing administrative commands.

3. Optionally create the asynchronous command handler for listening to command notifications.

Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.

4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Use the command manager that you created in a previous step to create and set up the command that exports an asset.

The command name is `exportAsset`. The `assetID` and `filename` parameters are required parameters to specify the asset to export and the file name and directory where the asset is exported.

6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to run the command that exports an asset.
You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.
8. Check the command result when the command completes.
When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, the asset is exported.

Example

The following example shows how to export an asset from a business-level application based on the previous steps.

Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class ExportAsset {

    public static void main(String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.

            String host = "localhost";
            String port = "8880"; // Change to your port number if it is
                                // not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create the command manager.
            CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);
```

```

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager:
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the following listener with
// null for the AsyncCommandClient object.
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create command that exports the asset.
String cmdName = "exportAsset";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Export as asset
// using the session created.
System.out.println("\nCreated " + cmdName);

// (required) Set the assetID parameter to the composition
// unit that you are exporting.
// Examples of valid formats for the assetID parameter are:
// - aName
// - assetname=aName
// - WebSphere:assetname=aName
// This parameter accepts an incomplete ID as long as
// the incomplete ID can resolve to a unique asset
// within the business-level application.
String assetID = "test5.zip";
cmd.setParameter("assetID", assetID);

System.out.println("\nSet assetID parameter to "
+ cmd.getParameter("assetID"));

// Set the file name for the asset to be exported. Use a
// fully qualified path name. An existing file with the specified
// name will be overwritten.
DownloadFile filename = new DownloadFile("/assets/asset1.zip");

cmd.setParameter("filename", filename);

System.out.println("\nSet filename parameter to "
+ cmd.getParameter("filename"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
"parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +

```

```

        "asyncCmdClientHelper.processCommandParameters(cmd).");
        th.printStackTrace();
        System.exit(-1);
    }

    // Call the asynchronous command client to run the command.
    asyncCmdClientHelper.execute(cmd);
    System.out.println("\nCompleted running of the command");

    // Check the command result.
    CommandResult result = cmd.getCommandResult();
    if (result != null) {
        if (result.isSuccessful()) {
            System.out.println("\nCommand ran successfully "
                + "with result\n" + result.getResult());
        }
        else {
            System.out.println("\nCommand ran with " +
                "Exception");
            result.getException().printStackTrace();
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}
}

```

What to do next

You can import the asset to another session. You can complete other tasks associated with assets, such as listing assets, and editing assets.

Listing assets using programming

You can list the assets that have been imported so that you can do further asset administration, such as deleting or exporting assets. An asset represents at least one binary file that implements business logic.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

You can list assets using programming, the administrative console, or the wsadmin tool.

About this task

You can list assets using programming, the administrative console, or the wsadmin tool. This topic describes how to list assets using programming.

When you list assets, all the assets are listed unless you set the `assetID` to specify the asset that you want to list. You can optionally include deployable units or a description of the assets when you list the assets. After you list the assets, you can use the information to do further administration, such as deleting or exporting assets.

Perform the following tasks to list assets using programming.

1. Connect to the application server.

The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.

2. Create the command manager.

The command manager provides the functionality to create a new administrative command or query existing administrative commands.

3. Optionally create the asynchronous command handler for listening to command notifications.

Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.

4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Use the command manager that you created in a previous step to create and set up the command that lists assets.

The command name is `listAssets`. You can optionally set the `assetID` parameter to query for assets that match the ID. You can also optionally set the `includeDescription` parameter and the `includeDeplUnit` parameter to include the display of the asset description and its deployable units.

6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to list the asset.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, a list of assets is displayed.

Example

The following example shows how to list the assets based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
```



```

import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class ListAssets {

    public static void main(String[] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.

            String host = "localhost";
            String port = "8880"; // Change to your port number if
                // it is not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create the command manager.
            CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

            // Comment out the previous lines to create a client command
            // manager if you are using a local command manager.
            // Uncomment the following line to create a local command
            // manager:
            //
            // CommandMgr cmdMgr = CommandMgr.getCommandMgr();
            System.out.println("\nCreated command manager");

            // Optionally create an asynchronous command handler.
            // Comment out the following line if no further handling
            // of command notification is required.
            AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

            // Create an asynchronous command client.

            // Set up the session.
            String id = Long.toHexString(System.currentTimeMillis());
            String user = "content" + id;
            Session session = new Session(user, true);

            // If no command handler is used, replace listener with
            // null for the AsyncCommandClient object.
            AsyncCommandClient asyncCmdClientHelper = new
                AsyncCommandClient(session, listener);
            System.out.println("\nCreated async command client");

            // Create the command that lists the assets.
            String cmdName = "listAssets";
            AdminCommand cmd = cmdMgr.createCommand(cmdName);
            cmd.setConfigSession(session); // list all the assets
                // using the session created.

```

```

System.out.println("\nCreated " + cmdName);

// Optionally set the assetID parameter.
// Uncomment the following code to set the assetID parameter to
// only list the asset with the ID specified, otherwise all
// assets are listed. Change the assetID parameter according to your
// scenario.
// Examples of valid formats for the assetID parameter are:
// - aName
// - assetname=aName
// - WebSphere:assetname=aName
// All assets that match the ID specification are listed.
// The ID must include at least the asset name.
// String assetID = "asset1.zip";
// cmd.setParameter("assetID", assetID);

//System.out.println("\nSet assetID parameter to "
//                    + cmd.getParameter("assetID"));

// Optionally include a description by setting
// the includeDescription parameter to true or false.
String includeDescription = "true";
cmd.setParameter("includeDescription", includeDescription);

System.out.println("\nSet includeDescription parameter to "
                  + cmd.getParameter("includeDescription"));

// Optionally include deployable units by setting
// the includeDeplUnit parameter to true or false.
String includeDeplUnit = "false";
cmd.setParameter("includeDeplUnit", includeDeplUnit);

System.out.println("\nSet includeDeplUnit parameter to "
                  + cmd.getParameter("includeDeplUnit"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
                      "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
                      "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Run the command to list assets.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of the command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
                          + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
                          "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

```

```

    '}'
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

You can complete other tasks associated with assets, such as deleting, editing, and exporting assets.

Listing composition units using programming

You can list the composition units for a specific business-level application so that you can complete further composition unit administration, such as deleting or adding composition units. A composition unit is typically created from a business-level application or an asset and contains configuration information that makes the asset runnable.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

Before you can list composition units, you must have imported an asset, created an empty business-level application, and added a composition unit to the business-level application.

About this task

You can list composition units using programming, the administrative console, or the wsadmin tool. This topic describes how to list composition units using programming.

You must provide the blaID parameter to specify the business-level application to list the composition unit. When you list composition units for a business-level application, you can optionally list the type for each composition unit and the description for each composition unit that has a description. You can use the list to complete further administration, such as deleting or exporting composition units.

Perform the following tasks to list composition units for a business-level application using programming.

1. Connect to the application server.

The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.

The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.

Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.

4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Use the command manager that you created in a previous step to create and set up the command that lists composition units.

The command name is `listCompUnits`. The `blalD` parameter is a required parameter that you use to specify the business-level application to list the composition units. You can optionally set the `includeDescription` parameter to display the composition unit descriptions. You can also optionally set the `includeType` parameter to display the composition unit types.

6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to list the composition units.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, a list of composition units for a business-level application is displayed.

Example

The following example shows how to list the composition units of a specific business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class ListCompUnits {

    public static void main(String[] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
```

```

// to get the soapClient soap client if you use the local
// command manager.

String host = "localhost";
String port = "8880"; // Change to your port number if
                    // it is not 8880.

Properties config = new Properties();
config.put(AdminClient.CONNECTOR_HOST, host);
config.put(AdminClient.CONNECTOR_PORT, port);
config.put(AdminClient.CONNECTOR_TYPE,
          AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println("Config: " + config);
AdminClient soapClient =
    AdminClientFactory.createAdminClient(config);

// Create the command manager.
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager:
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the listener with
// null for the AsyncCommandClient object.
AsyncCommandClient asyncCmdClientHelper = new
    AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that lists the composition units.
String cmdName = "listCompUnits";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // List all the composition units
                              // for the business-level application
                              // with this session ID.
System.out.println("\nCreated " + cmdName);

// Set the blaID parameter to the business-level application
// whose composition units are listing.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter accepts an incomplete ID as long as the incomplete
// ID can resolve to a unique business-level application.
String blaID = "bla1";
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
    + cmd.getParameter("blaID"));

```

```

// Optionally include descriptions for each composition unit
// that has a description by setting
// the includeDescription parameter to true or false.
String includeDescription = "true";
cmd.setParameter("includeDescription", includeDescription);

System.out.println("\nSet includeDescription parameter to "
    + cmd.getParameter("includeDescription"));
// Optionally include types for each composition unit
// by setting the includeType parameter to true or false.
String includeType = "true";
cmd.setParameter("includeType", includeType);

System.out.println("\nSet includeType parameter to "
    + cmd.getParameter("includeType"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Run the command to list the composition units.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

Now that you have listed the composition units for a business-level application, you can complete other tasks associated with composition units, such as adding or deleting composition units.

Listing business-level applications using programming

You can list the business-level applications of a session so that you can complete further business-level application administration such as deleting a business-level application. A business-level application is an administrative model that captures the definition of an enterprise-level application so that you can perform specific business functions, such as accounting.

Before you begin

Before you can list business-level applications of a session, you must have created an empty business-level application.

About this task

You can list business-level applications of a session using programming, the administrative console, or the wsadmin tool. This topic describes how to list business-level applications using programming.

List all the business-level applications of a session unless you set the `blatID` parameter to specify the business-level application that you want to list. You can optionally list the business-level applications with a description for those that have a description if you set the `includeDescription` parameter to `true`. After you list the business-level applications, you can use the information to do further administration, such as starting or deleting business-level applications.

Perform the following tasks to list business-level applications of a session using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager that you created in a previous step to create and set up the command that lists business-level applications of a session.
The command name is `listBLAs`. You can optionally set the `blatID` parameter to query for business-level applications that match the ID. You can optionally set the `includeDescription` parameter to display the business-level application descriptions.
6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.
The command framework asynchronous command model requires this call.
7. Call the asynchronous command client to list the business-level applications of a session.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, a list of business-level applications for a session is displayed.

Example

The following example shows how to list the business-level applications of a session based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class ListBLAs {

    public static void main(String[] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.

            String host = "localhost";
            String port = "8880"; // Change to your port number if
                                // it is not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                    AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create the command manager.
            CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

            // Comment out the previous lines to create a client command
            // manager if you are using a local command manager.
            // Uncomment the following line to create a local command
            // manager:
            //
```



```

// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace listener with
// null for the AsyncCommandClient object.
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that lists the business-level applications.
String cmdName = "listBLAs";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // list all the business-level applications
                                // using the session created.

System.out.println("\nCreated " + cmdName);

// Optionally set the blaID parameter.
// Uncomment the following code to set the blaID parameter to
// only list the business-level applications with the ID specified. Otherwise all
// business-level applications are listed. Change the blaID parameter according
// to your scenario.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// All business-level applications that match the ID specification
// are listed. The ID must include at least the business-level
// application name.
// String blaID = "bla1";
// cmd.setParameter("blaID", blaID);

//System.out.println("\nSet blaID parameter to "
//                    + cmd.getParameter("blaID"));

// Optionally include a description by setting
// the includeDescription parameter to true instead of false.
String includeDescription = "true";
cmd.setParameter("includeDescription", includeDescription);

System.out.println("\nSet includeDescription parameter to "
                    + cmd.getParameter("includeDescription"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
                        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
                        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

```

```

// Run the command to list business-level applications.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of the command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

You can complete other tasks associated with business-level applications, such as deleting, starting, or stopping business-level applications.

Editing a composition unit using programming

You can edit the configuration information in a composition unit of a business-level application if, for example, you want to change certain modules in the composition unit that are configured to run in specific targets. A composition unit is typically created from a business-level application or an asset and contains configuration information that makes the asset runnable.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

Before you can edit a composition unit of a business-level application, you must have created an empty business-level application, imported an asset, and added a composition unit to the business-level application.

About this task

You can edit a composition unit of a business-level application using programming, the administrative console, or the wsadmin tool. This topic describes how to edit a composition unit of a business-level application using programming.

You must provide the blaID and cuID parameters to specify the composition unit of the business-level application that you are editing.

Perform the following tasks to edit a composition unit of a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager that you created in a previous step to create and set up the command that edits a composition unit of a business-level application.
The command name is editCompUnit. Use the required blaID and cuID parameters to specify the composition unit of the business-level application that you are editing.
6. Call the processCommandParameters method in the asynchronous command client to process the command parameters.
The command framework asynchronous command model requires this call.
7. Set up the command step parameters.
You can edit various composition unit information through steps. The CUOptions step contains data about the composition unit such as its description, starting weight, and start and restart behavior. The MapTargets step contains target information about where to deploy the composition unit. The RelationshipOptions step contains shared library composition units on which this composition unit has a dependency. The ActivationPlanOptions step allows you to change runtime components for each deployable unit. You can edit parameters in these steps.
8. Call the asynchronous command client to run the command that edits a composition unit of a business-level application.
You might have created an asynchronous command handler to implement the AsyncCommandHandlerIF interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.
9. Check the command result when the command completes.
When the command finishes running, control is returned to the caller. You can then check the result by calling the command.getCommandResult method.

Results

After you successfully run the code, the composition unit of a business-level application is edited.

Example

The following example shows how to edit a composition unit of a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class EditCompUnit {

    public static void main(String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.

            String host = "localhost";
            String port = "8880"; // Change to your port number if it is
                // not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create the command manager.
            CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

            // Comment out the previous lines to create a client command
            // manager if you are using a local command manager.
            // Uncomment the following line to create a local command
            // manager:
            // CommandMgr cmdMgr = CommandMgr.getCommandMgr();
            System.out.println("\nCreated command manager");

            // Optionally create an asynchronous command handler.
            // Comment out the following line if no further handling
            // of command notification is required:
            AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

            // Create an asynchronous command client.
```

```

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the listener with
// null for the following AsyncCommandClient object:
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that edits the composition unit.
String cmdName = "editCompUnit";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Edit a certain composition
// unit of a business-level using the session created.
System.out.println("\nCreated " + cmdName);

// Set the blaID parameter.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter accepts an incomplete ID as long as the
// incomplete ID can resolve to a unique business-level application.
String blaID = "bla1";
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
+ cmd.getParameter("blaID"));

// Set the cuID parameter.
// Examples of valid formats for the cuID parameter are:
// - name
// - cuname=name
// - WebSphere:cuname=name
// This parameter accepts an incomplete ID as long as the
// incomplete ID can resolve to a unique composition unit
// within the business-level application.
String cuID = "cu1";
cmd.setParameter("cuID", cuID);

System.out.println("\nSet cuID parameter to "
+ cmd.getParameter("cuID"));

// Call the asynchronous client helper to process the command parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Set up the step parameters for the CUOptions step.
// The CUOptions step contains the following arguments that can be edited:
// description - description for the composition unit
// startingWeight - starting weight for the composition unit
//                  within the business-level application.
// startedOnDistributed - to start composition unit upon distribution
//                       to target nodes.
// Valid values are true, false.
// restartBehaviorOnUpdate - restart behavior for the composition

```

```

//          unit when the composition unit is updated.
// Valid values are DEFAULT, ALL, NONE
String stepName = "CUOptions";
CommandStep step = ((TaskCommand) cmd).gotoStep(stepName);

// Composition Unit description:
String description = "cul description changed in editCompUnit";

for(int i = 0; i < step.getNumberOfRows(); i++) {
    // Use the following code to change the composition unit step parameters
    // of the CUOptions step. Change your set of step parameters
    // as required by your scenario.

    // For example, set the description.
    step.setParameter("description", description, i);
    System.out.println("\nSet description parameter to " +
        step.getParameter("description", i));
}

// Set up the step parameters for the MapTargets step
stepName = "MapTargets";
step = ((TaskCommand) cmd).gotoStep(stepName);

// In this step the server parameter is required.
// server - target(s) to deploy the composition unit. The default is server1.
//      To add an additional target to the existing
//      target, add a prefix to the target with a "+". To
//      delete an existing target, add a prefix to the
//      target with a "#". To replace the existing
//      target, use the regular syntax as in the addCompUnit example.
// Example: server = "#server1+server2";
String server = "server1";

for(int i = 0; i < step.getNumberOfRows(); i++) {
    // Use the following code to set the server parameter of the MapTargets step.
    // Change your set of step parameters as required by your
    // scenario.

    // For example, set the server.
    step.setParameter("server", server, i);
    System.out.println("\nSet server parameter to " +
        step.getParameter("server", i));
}

// If the RelationshipOptions step is available, the selected
// deployable units of the source asset of the "primary" composition
// unit (that is, the composition unit being added) have dependencies
// on other assets for which there are matching "secondary" composition
// units in the business-level application. The RelationshipOptions step is much like
// CreateAuxCUOptions except that the required secondary composition
// units already exist. Also, each RelationshipOptions row maps one
// deployable unit to one or more secondary composition units, whereas,
// each CreateAuxCUOptions row maps one deployable unit to one
// asset dependency.
//
// Each RelationshipOptions row corresponds to one deployable unit
// with one or more dependency relationships and consists of
// parameter values for the dependency relationships. Some parameters
// are read-only and some of them are editable. To edit parameter
// values, use the same approach as that used to edit parameter values
// in the CUOptions step.
//
// The parameters for this step include:
//
//   deplUnit - The name of the deployable unit which has the
//              dependency. (Read-only.)
//   relationship - Composition unit dependencies in the form of a

```

```

//          list of composition unit IDs. Composition unit
//          IDs are separated by a "plus" sign (+). Each ID
//          can be fully or partially formed as shown with the
//          following examples:
//          WebSphere:cuname=SharedLib1.jar
//          WebSphere:cuname=SharedLib.jar
//          SharedLib.jar
// matchTarget - Specifies whether the server target for the secondary
// composition units are to match the server target for
// the primary composition unit. The default value
// is "true". If the value is set to "false", the
// secondary composition unit will be created with no
// target. The target on the secondary composition unit
// can be set at a later time with the editCompUnit
// command.
// for(int i = 0; i < step.getNumberOfRows(); i++) {
//     // Use the following if statement to set the relationship and matchTarget parameters
//     // of the RelationshipOptions step. Change your set of
//     // step parameters as required by your scenario.

//     // Uncomment the following code to match the deplUnit and then set
//     // the relationship differently.
//     //String deplUnit = (String) step.getParameter("deplUnit",
//     //                                             i);

//     //if (deplUnit.equals("a1.jar") {
//         // For example, change the relationship for the a1.jar file.
//         //step.setParameter("relationship", relationship, i);
//         //System.out.println("\nSet relationship parameter " +
//         //                    "to " + step.getParameter("relationship", i));

//         // For example, change matchTarget.
//         //step.setParameter("matchTarget", matchTarget, i);
//         //System.out.println("\nSet matchTarget parameter to "+
//         //                    step.getParameter("matchTarget", i));
//     //}
// }

// The addCompUnit command contains thr ActivationPlanOptions step.
// The user can set the ActivationPlanOptions step parameters similar to
// the step parameters for the CUOptions step in the previous examples.
// The arguments for this step include:
// deplUnit - deployable unit URI (read only parameter)
// activationPlan - specifies a list of runtime components in the
// format of specname=xxxx

// Run the command command to edit the composition unit.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of the command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

```

```

}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

After you edit the composition unit, you can run the updated business-level application.

Editing an asset using programming

You can edit the information of an asset such as its destination location, its relationship with other assets, and so on. An asset represents at least one binary file that implements business logic.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

Before you can edit an asset, you must have imported an asset.

You can edit an asset of a business-level application using programming, the administrative console, or the wsadmin tool.

About this task

You can edit an asset of a business-level application using programming, the administrative console, or the wsadmin tool. This topic describes how to edit an asset of a business-level application using programming.

You must provide the assetID parameter to specify the asset that you are editing.

Perform the following tasks to edit an asset of a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Use the command manager that you created in a previous step to create and set up the command that edits an asset.

The command name is `editAsset`. The `assetID` parameter is a required parameter to specify the asset that you are editing.

6. Call the asynchronous command client to process the command parameters.
7. Set up the command step parameters.

The `AssetOptions` step contains data about the asset such as its description, file permission, and relationship with other assets. You can edit various parameters in the `AssetOptions` step.

8. Call the asynchronous command client to run the command that edits an asset of a business-level application.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

9. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, the asset of a business-level application is edited.

Example

The following example shows how to edit an asset of a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class EditAsset {

    public static void main(String [] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager.

            // Comment out the lines to and including get the
            // soapClient soap client if you use the local command manager.
            // Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
```

```

// soapClient);
// to get the soapClient soap client if you use the
// local command manager.
String host = "localhost";
String port = "8880"; // Change to your port number if it is
// not 8880.

Properties config = new Properties();
config.put(AdminClient.CONNECTOR_HOST, host);
config.put(AdminClient.CONNECTOR_PORT, port);
config.put(AdminClient.CONNECTOR_TYPE,
    AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println("Config: " + config);
AdminClient soapClient =
    AdminClientFactory.createAdminClient(config);

// Create the command manager:
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager.
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create async command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the listener with
// null for the following AsyncCommandClient object.
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that edits the asset.
String cmdName = "editAsset";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Edit an imported asset
// using the session created.
System.out.println("\nCreated " + cmdName);

// Set the assetID parameter
// Examples of valid formats for the assetID parameter are:
// - aName
// - assetname=aName
// - WebSphere:assetname=aName
// This parameter accepts an incomplete ID as long as
// the incomplete ID can resolve to a unique asset.
String assetID = "asset1.zip";
cmd.setParameter("assetID", assetID);

System.out.println("\nSet assetID parameter to "
    + cmd.getParameter("assetID"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
}

```

```

        System.out.println("\nCompleted process command " +
            "parameters");
    } catch (Throwable th) {
        System.out.println("Failed from " +
            "asyncCmdClientHelper.processCommandParameters(cmd).");
        th.printStackTrace();
        System.exit(-1);
    }

    // Set up the step parameters for the AssetOptions step.
    String stepName = "AssetOptions";
    CommandStep step = ((TaskCommand) cmd).gotoStep(stepName);

    // Asset description:
    String description = "asset for testing";

    // destination of deployed asset
    String destinationUrl = "/myInstalledAssets/asset1.zip";
    // Asset type aspect:
    String typeAspect = "spec=sharedlib";

    // Asset validation:
    String validate = "yes";

    // Permission of files:
    String filePermission = ".*\\.dll=755";

    // Asset relationship:
    String relationship = "";

    for (int i = 0; i < step.getNumberOfRows(); i++) {
        // The following lines set the description and typeAspect
        // step parameters. There are other step parameters
        // in the AssetOptions step in the following comments. Change your set
        // of step parameters as required by your scenario.

        // For example, set description
        step.setParameter("description", description, i);
        System.out.println("\nSet description parameter to " +
            step.getParameter("description", i));

        // For example, set the typeAspect parameter.
        // Format of a typeAspect is:
        // WebSphere:spec=xxx,version=n.n+
        // WebSphere:spec=xxx,version=n.n
        step.setParameter("typeAspect", typeAspect, i);
        System.out.println("\nGet typeAspect: " +
            step.getParameter("typeAspect", i));

        // For example, set the destination parameter.
        step.setParameter("destination", destination, i);
        System.out.println("\nSet destination parameter to " +
            step.getParameter("destination", i));

        // For example, set the validate parameter.
        step.setParameter("validate", validate, i);
        System.out.println("\nSet validate parameter to " +
            step.getParameter("validate", i));

        // For example, set the filePermission parameter.
        step.setParameter("filePermission", filePermission, i);
        System.out.println("\nSet filePermission parameter to " +
            step.getParameter("filePermission", i));

        // For example, set relationship.
        step.setParameter("relationship", relationship, i);
    }

```

```

        System.out.println("\nSet relationship parameter to " +
            step.getParameter("relationship", i));
    }

    // Run the command to edit the asset.
    asyncCmdClientHelper.execute(cmd);
    System.out.println("\nCompleted running of command");

    // Check the command result.
    CommandResult result = cmd.getCommandResult();
    if (result != null) {
        if (result.isSuccessful()) {
            System.out.println("\nCommand ran successfully "
                + "with result\n" + result.getResult());
        }
        else {
            System.out.println("\nCommand ran with " +
                "Exception");
            result.getException().printStackTrace();
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}
}

```

```

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

After you edit the asset, you can add the asset as a composition unit to a business-level application, or export the asset.

Editing a business-level application using programming

You can edit the information of a business-level application such as its description. A business-level application is an administrative model that captures the entire definition of an enterprise-level application.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

Before you can edit a business-level application, you must have created a business-level application.

You can edit a business-level application using programming, the administrative console, or the wsadmin tool.

About this task

You must provide the `blaiD` parameter to specify the business-level application that you are editing.

Perform the following tasks to edit a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager that you created in a previous step to create the command that edits a business-level application.
The command name is `editBLA`. Use the required `blaiD` parameter to specify the business-level application that you are editing.
6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.
The command framework asynchronous command model requires this call.
7. Set up the command step parameter by setting the description parameter.
The `BLAOptions` step contains a description for the business-level application. You can edit the description parameter in the `BLAOptions` step.
8. Call the asynchronous command client to run the command to edit a business-level application.
You could have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.
9. Check the command result when the command completes.
When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, the business-level application is edited.

Example

The following example shows how to edit a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
```

```

import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class EditBLA {

    public static void main(String[] args) {
        try {

            // Connect to the application server.
            // This step is optional if you use the local command manager.
            // Comment out the following lines to get the soapClient SOAP client if
            // you are going to use the local command manager. You would
            // comment out the lines to and including
            // CommandMgr cmdMgr =
            // CommandMgr.getClientCommandMgr(soapClient);

            String host = "localhost"; // Change to your host if it is not localhost.
            String port = "8880"; // Change to your port number if it is not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create command manager.
            CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

            // Comment out the previous lines to create a client command
            // manager if you are using a local command manager.
            // Uncomment the following line to create a local command
            // manager.
            //
            // CommandMgr cmdMgr = CommandMgr.getCommandMgr();

            System.out.println("\nCreated command manager");

            // Optionally create an asynchronous command handler.
            // Comment out the following line if no further handling
            // of command notification is required.
            AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

            // Create an asynchronous command client.

            // Set up the session.
            // This example creates a new session. You can replace the
            // code below to use an existing session that has been
            // created.
            String id = Long.toHexString(System.currentTimeMillis());
            String user = "content" + id;
            Session session = new Session(user, true);

            // If no command handler is used, replace the listener with
            // null for the following AsyncCommandClient object.
            AsyncCommandClient asyncCmdClientHelper = new
                AsyncCommandClient(session, listener);
            System.out.println("\nCreated async command client");
        }
    }
}

```

```

// Create the command that edits the business-level application.
String cmdName = "editBLA";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Edit an existing business-level
                                // application using the session
                                // created.
System.out.println("\nCreated " + cmdName);

// Set the blaID parameter (required).
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter accepts an incomplete ID as long as the incomplete
// ID can resolve to a unique business-level application.
String blaID = "bla1"; // Replace bla1 with your value of the blaID.
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
    + cmd.getParameter("blaID"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Throwing an exception from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Set up the step parameters for the BLAOptions step.
// The only step parameter you can edit is description.
String stepName = "BLAOptions";
CommandStep step = ((TaskCommand) cmd).gotoStep(stepName);

// Edit the business-level application description.
String description = "bla for testing"; // Replace with your value.

for (int i = 0; i < step.getNumberOfRows(); i++) {
    // The following lines set the description
    // step parameter.
    step.setParameter("description", description, i);
    System.out.println("\nSet description parameter to " +
        step.getParameter("description", i));
}

// Run the command to edit the business-level application.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted command execution");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand executed successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand executed with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
}

```

```

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification.
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

After you edit the business-level application, you can continue administration of business-level applications. You can do such things as start and stop a business-level application, delete a business-level application, add a composition unit to a business-level application, and so on.

Updating an asset using programming

You can update an asset by adding, deleting, or updating a single file or Java Platform, Enterprise Edition (Java EE) module, or by merging multiple files or Java EE modules into an asset. You can also update an asset by replacing the entire asset.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interface documentation.

Before you can update an asset, you must have imported the asset.

You can update an asset using programming, the administrative console, or the wsadmin tool.

About this task

You must specify the `assetID` parameter of the asset that you are updating. In addition, you must specify the operation parameter. Whether or not you must specify the `contents` and `contentURI` parameters depends on the operation that you specify.

You modify one or more files or module files of an asset with this task. You also update the asset binary file, but do not update the composition units that the system deploys with this asset as a backing object.

Perform the following tasks to update an asset using programming.

1. Connect to the application server.

The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.

The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.

Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.

4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Create and set up the command that updates an asset.

- a. Set the parameter for the asset that you are updating.
- b. Set the operation parameter.
- c. Set the contents parameter unless the operation is set to delete.
- d. Set the contenturi parameter if the operation is set to add, update, or addupdate.

6. Call the processCommandParameters method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to run the command to update an asset.

You could have created an asynchronous command handler to implement the AsyncCommandHandlerIF interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the command.getCommandResult method.

Results

After you successfully run the code, the asset is updated.

Example

The following example shows how to update an asset based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class EditBLA {

    public static void main(String[] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local command manager.
            // Comment out the following lines to get soapClient soap client if
```

```

// you are going to use the local command manager.
// Comment out the lines to and including
// CommandMgr cmdMgr =
// CommandMgr.getClientCommandMgr(soapClient);

String host = "localhost"; // Change to your host if it is not localhost.
String port = "8880"; // Change to your port number if it is not 8880.

Properties config = new Properties();
config.put(AdminClient.CONNECTOR_HOST, host);
config.put(AdminClient.CONNECTOR_PORT, port);
config.put(AdminClient.CONNECTOR_TYPE,
    AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println("Config: " + config);
AdminClient soapClient =
    AdminClientFactory.createAdminClient(config);

// Create the command manager.
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager.
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();

System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required.
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
// This example creates a new session. You can replace the
// following code to use an existing session that has been
// created.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If you do not use the command handler, replace the listener with
// null for the following AsyncCommandClient object.
AsyncCommandClient asyncCmdClientHelper = new
    AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that updates the asset.
String cmdName = "updateAsset";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // Update an asset
                                // using the session
                                // created.
System.out.println("\nCreated " + cmdName);

// Set the required assetID parameter.
// Examples of valid formats for the assetID parameter:
// - aName
// - assetname=aName
// - WebSphere:assetname=aName
// This parameter accepts an incomplete ID as long as the
// incomplete ID can resolve to a unique asset within the
// business-level application.
String assetID = "asset1.zip"; // Replace asset1.zip with your

```

```

        // value of the assetID parameter.
cmd.setParameter("assetID", assetID);

System.out.println("\nSet assetID parameter to "
    + cmd.getParameter("assetID"));

// Set the required operation parameter.
// Possible operation values are add, addupdate, delete, merge,
// replace, and update.
// Use the add value to add a new file or Java EE module to the asset.
// Use the addupdate value to add a new file or Java EE module to the asset, or
// update an existing file or Java EE module.
// Use the delete value to delete an existing file or Java EE module in the asset.
// Use the merge value to provide a partial update with multiple
// additions, updates, or deletions.
// Use the replace value for a full update to replace all the contents.
// Use the update value to update an existing file or Java EE module in the asset.
String op = "add"; // Replace the add value with your operation value.
cmd.setParameter("operation", op);

System.out.println("\nSet operation parameter to "
    + cmd.getParameter("operation"));

// Set the contents parameter.
// This parameter is required unless the operation is set to
// delete.
String contents = "/assets/abc.txt"

cmd.setParameter("contents", contents);

System.out.println("\nSet contents parameter to "
    + cmd.getParameter("contents"));

// Set the contenturi parameter.
// This parameter is required for the
// add, addupdate, update, or delete operations.
String contenturi = "abc.txt"; // URI within the asset to
    // place the new file. Replace
    // with your value.
cmd.setParameter("contenturi", contenturi);

System.out.println("\nSet contenturi parameter to "
    + cmd.getParameter("contenturi"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Throwing an exception from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Run the command.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted command execution");

CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand executed successfully "
            + "with result\n" + result.getResult());
    }
}

```

```

        else {
            System.out.println("\nCommand executed with " +
                               "Exception");
            result.getException().printStackTrace();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification.
        System.out.println("\nEXAMPLE: notification received: " +
                           notification);
    }
}

```

What to do next

You can do other tasks associated with assets in business-level applications, such as adding or deleting other assets, listing assets, exporting assets, and so on.

Viewing a composition unit using programming

A composition unit is typically created from a business-level application or an asset and contains configuration information that makes the asset runnable. You can view the composition unit information so that you can complete other tasks associated with the composition unit such as editing an asset or delete a composition unit.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interfaces documentation.

Before you can view a composition unit of a business-level application, you must have created an empty business-level application, imported an asset into the business-level application, and added a composition unit to the business-level application.

About this task

You can view a composition unit using programming, the administrative console, or the wsadmin tool. This topic describes how to view a composition unit using programming.

You must provide the blaid and culd parameters to specify the composition unit of the business-level application that you are viewing. You can view configuration information of the composition unit of a business-level application. The configuration information identifies the asset from which the composition unit is created if the composition unit contains an asset. You can also view runtime targets on which the deployable units of the composition unit are to run.

Perform the following tasks to view a composition unit of a business-level application using programming.

1. Connect to the application server.

The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.

2. Create the command manager.

The command manager provides the functionality to create a new administrative command or query existing administrative commands.

3. Optionally create the asynchronous command handler for listening to command notifications.

Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.

4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Use the command manager that you created in a previous step to create and set up the command to view a composition unit.

The command name is `viewCompUnit`. Use the required `blalD` and `culD` parameters to specify the composition unit of the business-level application that you are viewing.

6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to run the command and view a composition unit.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, you can view the configuration information of a composition unit for a business-level application.

Example

The following example shows how to view a composition unit of a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class ViewCompUnit {
```

```

public static void main(String [] args) {

    try {
        // Connect to the application server.
        // This step is optional if you use the local
        // command manager. Comment out the lines to and including
        // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
        // soapClient);
        // to get the soapClient soap client if you use the local
        // command manager.
        String host = "localhost";
        String port = "8880"; // Change to your port number if it is
        // not 8880.

        Properties config = new Properties();
        config.put(AdminClient.CONNECTOR_HOST, host);
        config.put(AdminClient.CONNECTOR_PORT, port);
        config.put(AdminClient.CONNECTOR_TYPE,
            AdminClient.CONNECTOR_TYPE_SOAP);
        System.out.println("Config: " + config);
        AdminClient soapClient =
            AdminClientFactory.createAdminClient(config);

        // Create the command manager.
        CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

        // Comment out the previous lines to create a client command
        // manager if you are using a local command manager.
        // Uncomment the following line to create a local command
        // manager:
        //
        // CommandMgr cmdMgr = CommandMgr.getCommandMgr();
        System.out.println("\nCreated command manager");

        // Optionally create an asynchronous command handler.
        // Comment out the following line if no further handling
        // of command notification is required:
        AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

        // Create an asynchronous command client.

        // Set up the session.
        String id = Long.toHexString(System.currentTimeMillis());
        String user = "content" + id;
        Session session = new Session(user, true);

        // If no command handler is used, replace the following listener with
        // null for the AsyncCommandClient object.
        AsyncCommandClient asyncCmdClientHelper = new
        AsyncCommandClient(session, listener);
        System.out.println("\nCreated async command client");

        // Create the command that views the composition unit.
        String cmdName = "viewCompUnit";
        AdminCommand cmd = cmdMgr.createCommand(cmdName);
        cmd.setConfigSession(session); // View a certain composition
        // unit of a business-level
        // application using the session created.
        System.out.println("\nCreated " + cmdName);

        // (required) Set the blaID parameter.
        // Examples of valid formats for the blaID parameter are:
        // - bName
        // - blaname=bName
        // - WebSphere:blaname=bName
        // This parameter accepts an incomplete ID as long as the incomplete
        // ID can resolve to a unique business-level application.
    }
}

```

```

String blaID = "bla1";
cmd.setParameter("blaID", blaID);

System.out.println("\nSet blaID parameter to "
    + cmd.getParameter("blaID"));

// (required) Set the cuID parameter to the composition unit.
// The cuID parameter has the format of
// WebSphere:cuname=name. This parameter
// accepts an incomplete ID as long as the incomplete
// ID can resolve to a unique composition unit within the
// business-level application.
String cuID = "cu1";
cmd.setParameter("cuID", cuID);

System.out.println("\nSet cuID parameter to "
    + cmd.getParameter("cuID"));

// Call the asynchronous client helper to process parameters
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Call the asynchronous command client to run the command.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of the command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {
    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

You can use the information that you viewed about the composition unit to perform other tasks. For instance, you might edit the asset in the composition unit to make improvements to the asset. You might export the composition unit, and then import that composition unit into another business-level application.

Viewing an asset using programming

You can view the asset information so that you can complete other tasks associated with the asset, such as editing or exporting an asset. An asset represents at least one binary file that implements business logic.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interface documentation.

Before you can view an asset of a business-level application, you must have imported an asset.

About this task

You can view an asset using programming, the administrative console, or the wsadmin tool. This topic describes how to view an asset using programming.

You must provide the `assetID` parameter to specify the asset you are viewing. You can view configuration information of an asset, such as the destination location and relationships with other assets.

Perform the following tasks to view an asset of a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.
An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.
5. Use the command manager that you created in a previous step to create and set up the command to view an asset.
The command name is `viewAsset`. Use the required `assetID` parameter to specify the asset that you are viewing.
6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.
The command framework asynchronous command model requires this call.
7. Call the asynchronous command client to run the command and view an asset.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, you can view the configuration information of an asset.

Example

The following example shows how to view an asset based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class ViewAsset {

    public static void main(String [] args) {

        try {
            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.
            String host = "localhost";
            String port = "8880"; // Change to your port number if it is
            // not 8880.

            Properties config = new Properties();
            config.put(AdminClient.CONNECTOR_HOST, host);
            config.put(AdminClient.CONNECTOR_PORT, port);
            config.put(AdminClient.CONNECTOR_TYPE,
                AdminClient.CONNECTOR_TYPE_SOAP);
            System.out.println("Config: " + config);
            AdminClient soapClient =
                AdminClientFactory.createAdminClient(config);

            // Create the command manager.
            CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

            // Comment out the previous lines to create a client command
            // manager if you are using a local command manager.
            // Uncomment the following line to create a local command
            // manager:
            //
```

```

// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the following listener with
// null for the AsyncCommandClient object:
AsyncCommandClient asyncCmdClientHelper = new
AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command to view the asset.
String cmdName = "viewAsset";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // View a certain composition
                                // unit of a business-level application
                                // using the session created.
System.out.println("\nCreated " + cmdName);

// (required) Set the assetID parameter to the asset.
// Examples of valid formats for the assetID parameter:
// - aName
// - assetname=aName
// - WebSphere:assetname=aName
// This parameter accepts an incomplete ID as long as the
// incomplete ID can resolve to a unique asset.
String assetID = "asset1.zip";
cmd.setParameter("assetID", assetID);

System.out.println("\nSet assetID parameter to "
+ cmd.getParameter("assetID"));

// Call the asynchronous client helper to process parameters
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
"parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
"asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Call the asynchronous command client to run the command.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
+ "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +

```

```

        "Exception");
        result.getException().printStackTrace();
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {
    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

You can use the asset information that you viewed to perform other tasks. For instance, you might edit the asset to make improvements to the asset. You might export the asset and then import it into another configuration repository. You can then add the asset as a composition unit to a business-level application.

Viewing a business-level application using programming

You can view business-level application information such as the description so that you can do other tasks associated with the business-level application, such as editing the business-level application. A business-level application is an administrative model that captures the entire definition of an enterprise-level application.

Before you begin

This task assumes a basic familiarity with command framework programming. Read about command framework programming in the application programming interface documentation.

Before you can view a business-level application, you must have created the business-level application.

You can view a business-level application using programming, the administrative console, or the wsadmin tool.

About this task

You must provide the `blalD` parameter to specify the business-level application that you are viewing.

Perform the following tasks to view a business-level application using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.

Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.

4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Use the command manager that you created in a previous step to create and set up the command to view a business-level application.

The command name is viewBLA. Use the required blaID parameter to specify the business-level application that you are viewing.

6. Call the processCommandParameters method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Display the command step.

8. Call the asynchronous command client to run the command to view a business-level application.

You could have created an asynchronous command handler to implement the AsyncCommandHandlerIF interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

9. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the command.getCommandResult method.

Results

After you successfully run the code, you can view a business-level application.

Example

The following example shows how to view a business-level application based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.cmdframework.CommandStep;
import com.ibm.websphere.management.cmdframework.TaskCommand;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class EditBLA {

    public static void main(String[] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local command manager.
            // Comment out the following lines to get the soapClient soap client if
            // you are going to use the local command manager. You would
```

```

// comment out the lines to and including
// CommandMgr cmdMgr =
// CommandMgr.getClientCommandMgr(soapClient);

String host = "localhost"; // Change to your host if it is not localhost.
String port = "8880"; // Change to your port number if it is not 8880.

Properties config = new Properties();
config.put(AdminClient.CONNECTOR_HOST, host);
config.put(AdminClient.CONNECTOR_PORT, port);
config.put(AdminClient.CONNECTOR_TYPE,
    AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println("Config: " + config);
AdminClient soapClient =
    AdminClientFactory.createAdminClient(config);

// Create the command manager.
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager.
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();

System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required.
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
// This example creates a new session. You can replace the
// code below to use an existing session that has been
// created.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace the listener with
// null for the following AsyncCommandClient object.
AsyncCommandClient asyncCmdClientHelper = new
    AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command.
String cmdName = "viewBLA";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // View an existing
                                // business-level application
                                // using the session created.
System.out.println("\nCreated " + cmdName);

// Set the required blaID parameter.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter accepts an incomplete ID as long as the incomplete
// ID can resolve to a unique business-level application.
String blaID = "bla1"; // Replace the bla1 value with your value.
cmd.setParameter("blaID", blaID);

```

```

System.out.println("\nSet blaID parameter to "
    + cmd.getParameter("blaID"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
        "parameters");
} catch (Throwable th) {
    System.out.println("Throwing an exception from " +
        "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Display step data.
String[] stepNames = ((TaskCommand) cmd).listCommandSteps();
for (int i = 0; i < stepNames.length; i++) {

    // Get the step.
    CommandStep step =
        ((TaskCommand)cmd).gotoStep(stepNames[i]);

    List paramNames = step.listParameterName();

    System.out.println("----- Step: " + step.getName() +
        " -----");
    // Get the parameter values for each row.
    for (int j = 0; j < step.getNumberOfRows(); j++) {
        System.out.println(" Row " + j);

        for (int k = 0; k < paramNames.size(); k++)
            System.out.println("      " + paramNames.get(k) +
                ": " + step.getParameter(
                    (String) paramNames.get(k), j));

    }

}

// Run the command to view the business-level application.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted command execution");

CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand executed successfully "
            + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand executed with " +
            "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

```

```

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification.
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}

```

What to do next

You can use the information that you viewed about the business-level application to perform other tasks. You might edit the business-level application to make improvements to it. You might start and stop a business-level application, delete a business-level application, add a composition unit to a business-level application, and so on.

Listing control operations using programming

You can list the control operations of a business-level application or a composition unit for a session. You can use control operations, such as start or stop, to change or query the runtime environment of a business-level application or a composition unit.

Before you begin

Before you can list control operations of a business-level application or a composition unit for a session, you must have created an empty business-level application, imported an asset, and added a composition unit.

About this task

You can list control operations of a business-level application or a composition unit using programming, the administrative console, or the wsadmin tool. This topic describes how to list control operations using programming.

To list control operations for a business-level application of a session, provide a blaID parameter value, but no culD parameter value. To list control operations for a composition unit, specify both a blaID parameter value and a culD parameter value. To list all control operations for the specified business-level application or the specified composition unit, do not specify an opName parameter value. To list the details for a specific control operation, set the opName parameter value to the name of the operation to list. To list details of the control operation definition, set the long parameter to true.

Perform the following tasks to list control operations for a business-level application or a composition unit of a session using programming.

1. Connect to the application server.
The command framework allows the administrative command to be created and run with or without being connected to the application server. This step is optional if the application server is not running.
2. Create the command manager.
The command manager provides the functionality to create a new administrative command or query existing administrative commands.
3. Optionally create the asynchronous command handler for listening to command notifications.
Business-level application commands are implemented as asynchronous commands. To monitor the progress of the running command, you have to create an asynchronous command handler to receive notifications that the command generates.
4. Create the asynchronous command client.

An asynchronous command client provides a higher level interface to work with an asynchronous command. If you created an asynchronous command handler in the previous step, the handler is passed to the asynchronous command client. The asynchronous command client forwards the command notification to the handler and helps to control running of the command.

5. Create and set up the command that lists control operations of a business-level application or a composition unit of a session.
6. Call the `processCommandParameters` method in the asynchronous command client to process the command parameters.

The command framework asynchronous command model requires this call.

7. Call the asynchronous command client to list the control operations of a business-level application or a composition unit of a session.

You might have created an asynchronous command handler to implement the `AsyncCommandHandlerIF` interface class in a previous step. If you did, the asynchronous command client listens to command notifications and forwards the notifications to the handler. The handler performs any necessary actions while waiting for the command to complete.

8. Check the command result when the command completes.

When the command finishes running, control is returned to the caller. You can then check the result by calling the `command.getCommandResult` method.

Results

After you successfully run the code, a control operations of a business-level application or a composition unit for a session is displayed.

Example

The following example shows how to list the control operation of a business-level application or a composition unit of a session based on the previous steps. Some statements are split on multiple lines for printing purposes.

```
package com.ibm.ws.management.application.task;

import java.util.Properties;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.Session;
import com.ibm.websphere.management.cmdframework.AdminCommand;
import com.ibm.websphere.management.cmdframework.CommandMgr;
import com.ibm.websphere.management.cmdframework.CommandResult;
import com.ibm.websphere.management.async.client.AsyncCommandClient;

public class listControlOps {

    public static void main(String[] args) {

        try {

            // Connect to the application server.
            // This step is optional if you use the local
            // command manager. Comment out the lines to and including
            // CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(
            // soapClient);
            // to get the soapClient soap client if you use the local
            // command manager.
            String host = "localhost";
            String port = "8880"; // Change to your port number if
                                // it is not 8880.

            Properties config = new Properties();
```



```

config.put(AdminClient.CONNECTOR_HOST, host);
config.put(AdminClient.CONNECTOR_PORT, port);
config.put(AdminClient.CONNECTOR_TYPE,
           AdminClient.CONNECTOR_TYPE_SOAP);
System.out.println("Config: " + config);
AdminClient soapClient =
    AdminClientFactory.createAdminClient(config);

// Create the command manager.
CommandMgr cmdMgr = CommandMgr.getClientCommandMgr(soapClient);

// Comment out the previous lines to create a client command
// manager if you are using a local command manager.
// Uncomment the following line to create a local command
// manager:
//
// CommandMgr cmdMgr = CommandMgr.getCommandMgr();
System.out.println("\nCreated command manager");

// Optionally create an asynchronous command handler.
// Comment out the following line if no further handling
// of command notification is required:
AsyncCmdTaskHandler listener = new AsyncCmdTaskHandler();

// Create an asynchronous command client.

// Set up the session.
String id = Long.toHexString(System.currentTimeMillis());
String user = "content" + id;
Session session = new Session(user, true);

// If no command handler is used, replace listener with
// null for the AsyncCommandClient object.
AsyncCommandClient asyncCmdClientHelper = new
    AsyncCommandClient(session, listener);
System.out.println("\nCreated async command client");

// Create the command that lists the control operations.
String cmdName = "listControlOps";
AdminCommand cmd = cmdMgr.createCommand(cmdName);
cmd.setConfigSession(session); // List all the control operations
                               // using the session created.
System.out.println("\nCreated " + cmdName);

// Set the blaID parameter, which is required.
// The blaID is for either the business-level application whose control
// units you are listing or for the business-level application whose
// composition unit control operations you are listing.
// Change the blaID parameter according to your
// scenario.
// Examples of valid formats for the blaID parameter are:
// - bName
// - blaname=bName
// - WebSphere:blaname=bName
// This parameter accepts an incomplete ID as long as the incomplete
// ID can resolve to a unique business-level application.
// String blaID = "bla1";
// cmd.setParameter("blaID", blaID);

// System.out.println("\nSet blaID parameter to "
//                     + cmd.getParameter("blaID"));

// Optionally set the cuID parameter to the composition
// unit whose control operations you are listing.
// Examples of valid formats for the cuID parameter are:
// - name

```

```

// - cuname=name
// - WebSphere:cuname=name
// This parameter accepts an incomplete ID as long as the
// incomplete ID can resolve to a unique composition unit
// within the business-level application.
//
// String cuID = "test5.zip";
// cmd.setParameter("cuID", cuID);

// System.out.println("\nSet cuID parameter to "
//                    + cmd.getParameter("cuID"));

// Optionally set the opName parameter of the operation to list.
// String opName = "opName1";
// cmd.setParameter("opName", opName);

// System.out.println("\nSet opnameID parameter to "
//                    + cmd.getParameter("opName"));

// Optionally include details of the control operation definition
// by setting the long parameter to true.
// String long = "true";
// cmd.setParameter("long", long);

// System.out.println("\nSet long parameter to "
//                    + cmd.getParameter("long"));

// Call the asynchronous client helper to process parameters.
try {
    asyncCmdClientHelper.processCommandParameters(cmd);
    System.out.println("\nCompleted process command " +
                      "parameters");
} catch (Throwable th) {
    System.out.println("Failed from " +
                      "asyncCmdClientHelper.processCommandParameters(cmd).");
    th.printStackTrace();
    System.exit(-1);
}

// Run the command to list control operations.
asyncCmdClientHelper.execute(cmd);
System.out.println("\nCompleted running of command");

// Check the command result.
CommandResult result = cmd.getCommandResult();
if (result != null) {
    if (result.isSuccessful()) {
        System.out.println("\nCommand ran successfully "
                          + "with result\n" + result.getResult());
    }
    else {
        System.out.println("\nCommand ran with " +
                          "Exception");
        result.getException().printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

package com.ibm.ws.management.application.task;

import com.ibm.websphere.management.cmdframework.provider.CommandNotification;

```

```
import com.ibm.websphere.management.async.client.AsyncCommandHandlerIF;

public class AsyncCmdTaskHandler implements AsyncCommandHandlerIF {

    public void handleNotification(CommandNotification notification) {
        // Add your own code here to handle the received notification
        System.out.println("\nEXAMPLE: notification received: " +
            notification);
    }
}
```

What to do next

You can complete other tasks associated with business-level applications and composition units, such as deleting, starting, or stopping business-level applications or adding or exporting a composition unit.

Chapter 11. Troubleshooting deployment

When you are having problems deploying an application, perform some basic diagnostics and verify your system's configuration to solve the problem.

- Select the problem you are having with deploying or installing developed code for WebSphere Application Server.
 - Errors or problems deploying, installing, or promoting applications
- To troubleshoot other deployment issues, use the following resources.
 - For current information available from IBM Support on known problems and their resolution, see the IBM Support page.
 - IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.
 - If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting help from IBM.

Application deployment problems

You might encounter problems when deploying, installing, or promoting applications. This topic suggests ways to resolve the problems.

What kind of problem are you having?

- "I installed my application using the wsadmin tool, but the application does not display under Applications > Application Types > WebSphere enterprise applications" on page 482
- "Unable to save a deployed application" on page 482
- "WASX7015E error running wsadmin command \$AdminApp installInteractive or \$AdminApp install" on page 482
- "Cannot install a CMP or BMP entity bean in an EJB 3.0 module" on page 482
- "Data definition language (DDL) generated by an assembly tool throws SQL error on target platform" on page 483
- "Error message ADMA0004E: Validation error in task Specifying the Default Datasource for EJB Modules returned when installing application using the administrative console or the wsadmin tool" on page 483
- "Cannot load resource WEB-INF/ibm-web-bnd.xmi in archive file" on page 484
- ""Timeout!!!" error displays when attempting to install an enterprise application in the administrative console " on page 484
- "I get a NameNotFoundException message when deploying an application that contains an EJB module" on page 484
- "I get compilation errors and EJB deploy fails when installing an EJB JAR file generated for Version 5.x or earlier" on page 484
- "After installing the application onto a different machine, the application does not run" on page 485
- "A single file replaces all application files during application update" on page 485

Check the following first:

- Verify that the logical name that you have specified to appear on the console for your application, enterprise bean module or other resource does not contain invalid characters such as these: - / \ : * ? " < > |.
- If the application was installed using the wsadmin \$AdminApp install command with the **-local** flag, restart the server or rerun the command without the `-local` flag.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, check to see if the problem is identified and documented.

I installed my application using the wsadmin tool, but the application does not display under Applications > Application Types > WebSphere enterprise applications

The application might be installed but you have not saved the configuration:

1. Verify that the application subdirectory is located under the *app_server_root*/installedApps directory.
2. Run the \$AdminApp list command and verify that the application is not among those displayed.
 - In the bin directory, run the wsadmin.bat or wsadmin.sh command.
 - From the wsadmin prompt, enter \$AdminApp list and verify that the problem application is not among the items that display.
3. Reinstall your application using the wsadmin tool. Run the \$AdminConfig save command in the wsadmin tool before exiting.

Unable to save a deployed application

If you are unable to save a deployed application, the problem might be that too many files are opened, exceeding the limit of the operating system.

Only root has authority to adjust the maximum number of files for each process. Complete the following steps to modify the application to close files with disciplines:

1. After you open a file and complete your work, call the close method of the file to release the file handle back to the operating system.
2. Using the java.io.FileInputStream and the FileOutputStream classes as examples, you can invoke their close method to release any system resources that are associated with the stream.

WASX7015E error running wsadmin command \$AdminApp installInteractive or \$AdminApp install

This problem has two possible causes:

- If the full text of the error is similar to:

```
WASX7015E: Exception running command:
"$AdminApp installInteractive C:/Documents and Settings/
myUserName/Desktop/MyApp/myapp.ear";
exception information:
com.ibm.bsf.BSFException: error while
eval'ing Jacl expression: can't find method "installInteractive"
with 3 argument(s) for class
"com.ibm.ws.scripting.AdminAppClient"
```

The file and path name are incorrectly specified. In this case, since the path included spaces, it was interpreted as multiple parameters by the wsadmin program.

Enter the path of the .ear file correctly. In this case, by enclosing it in double quotes:

```
$AdminApp installInteractive "C:\Documents
and Settings\myUserName\Desktop\MyApps\myapp.ear"
```

- If the full text of the error is similar to:

```
WASX7015E: Exception running command: "$AdminApp installInteractive c:\MyApps\myapp.ear ";
exception information: com.ibm.ws.scripting.ScriptingException: WASX7115E:
Cannot read input file
"c:\WebSphere\AppServer\bin\MyAppsmyapp.ear"
```

The application path is incorrectly specified. In this case, you must use "forward-slash" (/) separators in the path.

Cannot install a CMP or BMP entity bean in an EJB 3.0 module

When installing an EJB 3.0 module that contains a container-managed persistence (CMP) or bean-managed persistence (BMP) entity bean, the installation fails.

The product does not support installation of applications that have a CMP or BMP entity bean packaged in an EJB 3.0 module. You must package CMP or BMP entity beans in an EJB 2.1 or earlier module.

To resolve this problem:

1. Package the CMP or BMP entity beans in EJB 2.1 or earlier modules.
2. Try installing your application with the EJB 2.1 or earlier modules.

Data definition language (DDL) generated by an assembly tool throws SQL error on target platform

If you receive SQL errors in attempting to execute data definition language (DDL) statements generated by an assembly tool on a different platform, for example if you are deploying a container-managed persistence (CMP) enterprise bean designed on Windows onto a UNIX operating system server, try the following actions:

- Browse the DDL statements for dependencies on specific user identifiers and passwords, and correct as necessary.
- Browse the DDL statements for dependencies on specific server names, and correct as necessary.
- Refer to the message reference of the vendor for causes and suggested actions regarding specific SQL errors. For IBM DB2, you can view the message references online at <http://www.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/index.d2w/report>.

If you receive the following error after executing a DDL file created on the Windows operating system or on operating systems such as AIX or Linux, the problem might come from a difference in file formats:

```
SQL0104N  An unexpected token "CREATE TABLE AGENT (COMM DOUBLE, PERCENT DOUBLE, P"
was found following "      ".  Expected tokens may include: " ".
SQLSTATE=42601
```

To resolve this problem:

- Use EDTF to edit the file.

Error message ADMA0004E: Validation error in task Specifying the Default Datasource for EJB Modules returned when installing application using the administrative console or the wsadmin tool

If you see the following error when trying to install an application through the administrative console or the wsadmin command prompt:

```
AppDeploymentException: [ADMA0014E: Validation failed.
ADMA0004E: Validation error in task Specifying the Default Datasource for
EJB Modules  JNDI name is not
specified for module beanameBean Jar with URI filename.jar,META-INF/ejb-jar.xml.
You have not specified the
data source for each CMP bean belonging to this module. Either specify the data
source for each CMP beans or
specify the default data source for the entire module.]
```

one possible cause is that, in WebSphere Application Server Version 4.0, it was mandatory to have a data source defined for each CMP bean in each JAR. In Version 5.0 and later releases, you can specify either a data source for a container-managed persistence (CMP) bean or a default data source for all CMP beans in the JAR file. Thus during installation interaction, such as the installation wizard in the administrative console, the data source fields are optional, but the validation performed at the end of the installation checks to see that at least one data source is specified.

To correct this problem, step through the installation again, and specify either a default data source or a data source for each CMP-type enterprise bean.

If you are using the wsadmin tool, use the `$AdminApp installInteractive filename` command to receive prompts for data sources during installation, or to provide them in a response file.

Specify data sources as an option to the `$AdminApp install` command.

Cannot load resource WEB-INF/ibm-web-bnd.xmi in archive file

The Web application `tmp.war` installs on WebSphere Application Server Versions 5.0 and 5.1, but fails on a WebSphere Application Server Version 6.0 or later server. The application fails to install because the `WEB-INF/ibm-web-bnd.xmi` file contains `xmi` tags that the underlying WCCM model no longer recognizes.

The following error messages display:

```
IWAE0007E Could not load resource "WEB-INF/ibm-web-bnd.xmi" in archive "tmp.war"
[2/24/05 14:53:10:297 CST] 000000bc SystemErr      R
AppDeploymentException:
com.ibm.etools.j2ee.commonarchivecore.exception.ResourceLoadException:
IWAE0007E Could not load resource "WEB-INF/ibm-web-bnd.xmi" in archive "tmp.war"
[2/24/05 14:53:10:297 CST] 000000bc SystemErr      R
com.ibm.etools.j2ee.commonarchivecore.exception.ResourceLoadException:
IWAE0007E Could not load resource "WEB-INF/ibm-web-bnd.xmi" in archive "tmp.war"
!Stack_trace_of_nested_exce!
com.ibm.etools.j2ee.exception.WrappedRuntimeException: Exception occurred loading
WEB-INF/ibm-web-bnd.xmi
!Stack_trace_of_nested_exce!
```

To work around this problem, remove the `xmi:type=EJBLocalRef` tag from the `ibm-web-bnd.xmi` file. Removing this tag does not affect the application because the tag was previously used for matching the cross document reference type. The application now works for the WebSphere Application Server Version 5.1 and later releases.

"Timeout!!!" error displays when attempting to install an enterprise application in the administrative console

This error can occur if you attempt to install an enterprise application that has not been deployed.

To correct this problem:

- Open the `file_name.ear` file in an assembly tool and then click **Deploy**. This action creates a file with a name like `Deployed_file_name.ear`.
- In the administrative console, install the deployed `.ear` file.

I get a NameNotFoundException message when deploying an application that contains an EJB module

If you specify that the EJB deployment tool be run during application installation and the installation fails with a `NameNotFoundException` message, ensure that the input JAR or EAR file does not contain source files. If there are source files in the input JAR or EAR file, the EJB deployment tools runs a rebuild before generating the deployment code.

To work around this problem, either remove the source files or include all dependent classes and resource files on the class path. Otherwise, the source files or the lack of access to dependent classes and resource files might cause problems during rebuilding of your application on the server.

I get compilation errors and EJB deploy fails when installing an EJB JAR file generated for Version 5.x or earlier

When installing an old application that uses EJB modules that were built to run on WebSphere Application Server Version 5.x or earlier, compilation errors result and EJB deploy fails. The EJB JAR file contains Java source for the old generated code. The old Java source was generated for Version 5.x or before but, when deployed to a WebSphere Application Server Version 6.x product, it is compiled using the Version 6.x runtime JAR files.

To work around this problem, remove all `.java` files from the application `.ear` file. After the Java source files are removed, you can deploy the application onto a server successfully.

After installing the application onto a different machine, the application does not run

If your application uses application level resources, its application level node information must be correct for the application to run as expected.

When you add application level resources to an application and deploy the application onto a machine, ensure that the application level node information is correct. Otherwise, when you install the application onto a different machine, it is installed to the wrong location and the application does not run as expected.

You can update the application level node information using an assembly tool. Update the `nodeName` from `deploymentTargets` of the `deployment.xml` file under `ibmconfig`. Also, ensure that `binariesURL` from `deployedObject` of the `deployment.xml` file has the correct path.

A single file replaces all application files during application update

If you select the **Replace or add a single file** option of the application update wizard and the currently deployed application consists of several files, specify the full path name of the file to be replaced or added for **Specify the path beginning with the installed application archive file to the file to be replaced or added**.

A full path name usually has the structure *directory_path/file_name* and resembles the following:

```
PriceChangeSession.jar/priceChangeSession/priceChangeSessionBean.class
```

Do not specify less than the full path name for **Specify the path beginning with the installed application archive file to the file to be replaced or added**. For example, do not specify only a directory path:

```
PriceChangeSession.jar/priceChangeSession
```

If you specify less than a full path name, all files in the directory of the currently deployed application might be replaced by the single new file that was specified under **Specify the path to the file**.

Application deployment troubleshooting tips

When you first test or run a deployed application, you might encounter problems.

Select the problem you are having with testing or the first run of deployed code for WebSphere Application Server:

- “Server startup problems” on page 512.
- “Application startup problems” on page 491.
- “Web resource is not displayed” on page 494.
- Data access problems.
- Enterprise bean cannot be accessed from a servlet, a JSP file, a stand-alone program, or another client.
- Application access problems.
- Access problems after enabling security.
- Security enablement followed by errors.
- Secure Sockets Layer errors.
- Application client sending SOAP request receives errors.
- “A client program does not work” on page 486.
- WebSphere MQ connection and queue connection factory creation errors.

You can use the following administrative console panels to inspect the configuration of your applications and JMS resources:

- For a view of the JMS resources for a given application, see the following panel: Messaging resources for this application.
- For a view of the applications and JMS resources for a given default messaging provider destination, see the following panel: Application resources for this destination.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting help from IBM.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the Must gather documents page for information to gather to send to IBM Support page.

A client program does not work

What kind of problem are you seeing?

ActiveX client fails to display ASP files, or WebSphere Application Server resources (JSP files, servlet, or HTML pages) or both

A possible cause of this problem is that both IIS for serving Active Server Pages (ASP) files and an HTTP server that supports WebSphere Application Server (such as IBM HTTP Server) are deployed on the same host. This deployment leads to misdirected HTTP traffic if both servers are listening on the same port (such as the default port 80).

To resolve this problem, either:

- Open the IIS administrative panel, and edit the properties of the default Web server to change the port number to a value other than 80
- Install IIS and the HTTP server on separate servers.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

Plants by WebSphere Catalog Manager (pbwsCatalogMgr) exceptions

When you federate a standalone server into a Deployment Manager cell, the bootstrap port number of the application server may change. This will cause the client to not be able to communicate with the server, thus causing an exception. The following scenario may cause an exception when you start Plants by WebSphere:

1. Install a standalone WebSphere Application Server.
2. Run the Plants by WebSphere example.
3. Create a Deployment Manager (DMGR) using the Profile Management tool or by using the **manageprofiles** command.
4. Federate the standalone WebSphere Application Server into a Deployment Manager cell using the **addNode** command.
5. Start **pbwsCatalogMgr**.

To avoid the exception, locate the new (changed) port number on the server and modify the client configuration to match the port number on the server.

1. Go to `was_server_root\profiles\your_server_name\config\cells\your_cell\nodes\your_node`.
 - a. Open the `serverindex.xml` file.
 - b. Locate the `BOOTSTRAP_ADDRESS` port number of the application server, for example 9810.

2. Assign this port number to the client to communicate with your newly-federated application server. Go to `was_client_root\bin` and edit the `setupClient.bat` file.
3. Locate the line 'SET SERVERPORTNUMBER' and set the value for it to 9810.
If you have security enabled, ensure that the bus security is also enabled and that a user is defined to the bus connector role before running `pbwsCatalogMgr`.
4. Restart the node agent and the application server.

The client is now properly set up to start **pbwsCatalogMgr**.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Application startup errors

Use this information for troubleshooting problems that occur when starting an application.

What kind of error do you see when you start an application?

- "HTTP server and Application Server are working separately, but requests are not passing from HTTP server to Application Server"
- "File serving problems" on page 488
- "Graphics do not appear in the JSP file or servlet output" on page 488
- "SRVE0026E: [Servlet Error]-[Unable to compile class for JSP file" on page 489
- "After modifying and saving a JSP file, the change does not show up in the browser (the old JSP file displays)" on page 490
- "Message like "Message: /jspname.jsp(9,0) Include: Mandatory attribute page missing" appears when attempting to browse JSP file" on page 490
- "The Java source generated from a JSP file is not retained in the temp directory (only the class file is found)" on page 490
- "The JSP Batch Compiler fails with the message "Enterprise Application [application name you typed in] not found."" on page 491
- "There is a translation problem with non-English browser input" on page 491
- "Scroll bars do not appear around items in the browser window" on page 491
- "Error "Page cannot be displayed... server not found or DNS error" appears when attempting to browse a JavaServer Pages (JSP) file using Internet Explorer" on page 491

HTTP server and Application Server are working separately, but requests are not passing from HTTP server to Application Server

If your HTTP server appears to be functioning correctly, and the Application Server also works on its own, but browser requests sent to the HTTP server for pages are not being served, a problem exists in the WebSphere Application Server plug-in.

In this case:

1. Determine whether the HTTP server is attempting to serve the requested resource itself, rather than forwarding it to the WebSphere Application Server.
 - a. Browse the HTTP server access log (*IHS install root/logs/access.log* for IBM HTTP Server). It might indicate that it could not find the file in its own document root directory.
 - b. Browse the plug-in log file as described below.
2. Refresh the `plugin-cfg.xml` file that determines which requests sent to the HTTP server are forwarded to the WebSphere Application Server, and to which Application Server.

Use the console to refresh this file:

- In the WebSphere Application Server administrative console, expand the Environment tree control.
- Click **Update WebSphere Plugin**.
- Stop and restart the HTTP server.

If you are using IBM HTTP Server for iSeries or Lotus Domino for iSeries, you do not need to restart the HTTP server.

- Retry the Web request.

If you have created a Web server definition to model your Web server instance, the file is located under *profile_root/config/cells/cell_name/nodes/Web_server_node_name/servers/Web_server_name*. If you have not, the file is located under *profile_root/config/cells*.

3. Browse the *plugin_install_root/logs/web_server_name/http_plugin.log* file for clues to the problem. Make sure the timestamps with the most recent plug-in information stanza, which is printed out when the plug-in is loaded, correspond to the time the Web server started.
4. Turn on plug-in tracing by setting the `LogLevel` attribute in the *plugin-cfg.xml* file to `Trace` and reloading the request. Browse the *plugin_install_root/logs/Web_server_name/http_plugin.log* file. You should be able to see the plug-in attempting to match the request URI with the various URI definitions for the routes in the *plugin-cfg.xml*. Check which rules the plug-in is not matching against and then figure out if you need to add additional ones. If you just recently installed the application you might need to manually regenerate the plug-in configuration to pick up the new URIs related to the new application.

For further details on troubleshooting plug-in-related problems, see Webserver plug-in troubleshooting tips located in the *Administering applications and their environment* PDF book.

File serving problems

If text output appears on your JSP- or servlet-supported Web page, but image files do not:

- Verify that your files are in the right place: the **document root** directory of your Web application. WebSphere Application Server follows the J2EE standard, which means that the document root is the *Web_module_name.war* directory of your deployed Web application.

Typically this directory will be found in the *profile_root/installedApps/nodename/appname.ear* directory or *profile_root/installedApps/nodename/appnameNetwork.ear* directory.

If the files are in a subdirectory of the document root, verify that the reference to the file reflects that. That is, if the *invoices.html* file is stored in *Windows* directory *Web_module_name.war\invoices*, then links from other pages in the Web application to display it should read "*invoices\invoices.html*", not "*invoices.html*".

- Verify that your Web application is configured to enable file serving (in other words, that it is enabled to display static resources like image and *.html* files):

1. View the file serving property of the hosting Web module by browsing the source *.war* file in an assembly tool. If necessary, update the property and redeploy the module. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

2. Edit the **fileServingEnabled** property in the deployed Web application *ibm-web-ext.xmi* configuration file.

The file typically is found in the *profile_root/config/cells/nodename* or *nodenameNetwork/applications/application_name/deployments/application_name/Web_module_name/web-inf* directory.

Graphics do not appear in the JSP file or servlet output

If text output appears on your JSP- or -servlet-supported Web page, but image files do not:

- Verify that your graphic files are in the right place: the **document root** directory of your Web application. WebSphere Application Server Version 5 follows the J2EE standard, which means that the document root is the *Web_module_name.war* directory of your deployed Web application.

Typically, this directory is found in the *profile_root/installedApps/nodename/appname.ear* directory or *profile_root/installedApps/nodename/appnameNetwork.ear* directory.

If the graphics files are in a subdirectory of the document root, verify that the reference to the graphic reflects that; for example, if the banner.gif file is stored in Windows directory *Web_module_name.war/images*, the tag to display it should read: ``, not ``.

- Verify that your Web application is configured to enable file serving (that is, display of static resources like image and .html files).
 1. View the file serving property of the hosting Web module by browsing the source .war file in an assembly tool. If necessary, update the property and redeploy the module. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
 2. Edit the **fileServingEnabled** property in the deployed Web application *ibm-web-ext.xmi* configuration file.

The file typically is found in the *profile_root/config/cells/nodename* or *nodenameNetwork/applications/application_name/deployments/application_name/Web_module_name/web-inf* directory.
 3. After completing the previous step:
 - In the administrative console, expand the **Environment** tree control .
 - Click **Update WebSphere Plugin**.
 - Stop and restart the HTTP server and retry the Web request.

SRVE0026E: [Servlet Error]-[Unable to compile class for JSP file

If this error appears in a browser when trying to access a new or modified .jsp file for the first time, the most likely cause is that the JSP file Java source failed (was incorrect) during the javac compilation phase.

Check the SystemErr.log file for a compiler error message, such as:

```
C:\WASROOT\temp\ ... test.war\_myJsp.java:14: \Duplicate variable declaration: int myInt was int myInt
int myInt = 122;
String myString = "number is 122";
static int myStaticInt=22;
int myInt=121;
    ^
```

Fix the problem in the JSP source file, save the source and request the JSP file again.

If this error occurs when trying to serve a JSP file that was copied from another system where it ran successfully, then there is something different about the new server environment that prevents the JSP file from running. Browse the text of the error for a statement like:

```
Undefined variable or class name: MyClass
```

This error indicates that a supporting class or jar file is not copied to the target server, or is not on the class path. Find the MyClass.class file, and place it on the Web module WEB-INF/classes directory, or place its containing .jar file in the Web module WEB-INF/lib directory.

Verify that the URL used to access the resource is correct by doing the following:

- For a JSP file, html file, or image file: **http://host_name/Web_module_context_root/subdir under doc root, if any/filename.ext**. The document root for a Web application is the *application_name.WAR* directory of the installed application.
 - For example, to access the myJsp.jsp file, located in *c:\WebSphere\ApplicationServer\installedApps\myEntApp.ear\myWebApp.war\invoices* on *myhost.mydomain.com*, and assuming the context root for the myWebApp Web module is *myApp*, the URL is *http://myhost.mydomain.com/myApp/invoices/myJsp.jsp*.
 - JSP serving is enabled by default. File serving for HTML and image files must be enabled as a property of the Web module, in an assembly tool, or by setting the **fileServingEnabled** property to **true** in the *ibm-web-ext.xmi* file of the installed Web application and restarting the application. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

- For servlets served by class name, the URL is `http://hostname/web_module_context_root/servlet/packageName.className`.
For example, to access `myCom.myServlet.class`, located in `profile_root/installedApps/myEntApp.ear/myWebApp.war/WEB-INF/classes`, and assuming the context root for the `myWebApp` module is `"myApp"`, the URL would be `http://myhost.mydomain.com/myApp/servlet/myCom.MyServlet`.
- Serving servlets by class name must be enabled as a property of the Web module, and is enabled by default. File serving for HTML and image files must be enabled as a property of the Web application, in an assembly tool, or by setting the `fileServingEnabled` property to `true` in the `ibm-web-ext.xmi` file of the installed Web application and restarting the application. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

Correct the URL in the "from" HTML file, servlet or JSP file. An HREF with no leading slash (/) inherits the calling resource context. For example:

- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to `"ServletB"` resolves to `"http://hostname/myapp/servlet/ServletB"`
- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to `"servlet/ServletB"` resolves to `"http://hostname/myapp/servlet/servlet/ServletB"` (an error)
- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to `"/ServletB"` resolves to `"http://hostname/ServletB"` (an error, if `ServletB` requires the same context root as `MyServlet`)

After modifying and saving a JSP file, the change does not show up in the browser (the old JSP file displays)

It is probable that the Web application is not configured for servlet reloading, or the reload interval is too high.

To correct this problem, in an assembly tool, check the **Reloading Enabled** flag and the **Reload Interval** value in the IBM Extensions for the Web module in question. Enable reloading, or if it is already enabled, then set the Reload Interval lower. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

Message like "Message: /jspname.jsp(9,0) Include: Mandatory attribute page missing" appears when attempting to browse JSP file

It is probable that the JSP file failed during the translation to Java phase. Specifically, a JSP directive, in this case an Include statement, was incorrect or referred to a file that could not be found.

To correct this problem, fix the problem in the JSP source, save the source and request the JSP file again.

The Java source generated from a JSP file is not retained in the temp directory (only the class file is found)

It is probable that the JSP processor is not configured to keep generated Java source.

In an assembly tool, check the **JSP Attributes** under **Assembly Property Extensions** for the Web module in question. Make sure the `keepgenerated` attribute is there and is set to `true`. If not, set this attribute and restart the Web application. To see the results of this operation, delete the class file from the temp directory to force the JSP processor to translate the JSP source into Java source again. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

The JSP Batch Compiler fails with the message "Enterprise Application [application name you typed in] not found."

It is probable that the full enterprise application path and name, starting with the .ear subdirectory that resides in the applications directory is expected as an argument to the JspBatchCompiler tool, not just the display name.

The directory path is *profile_root/config/cells/node_nameNetwork/applications*.

For example:

- "JspBatchCompiler -enterpriseapp.name sampleApp.ear/deployments/sampleApp" is correct, as opposed to
- "JspBatchCompiler -enterpriseapp.name sampleApp", which is incorrect.

There is a translation problem with non-English browser input

If non-English-character-set browser input cannot be translated after being read by a servlet or JSP file, ensure that the request parameters are encoded according to the expected character set before reading. For example, if the site is Chinese, the target .jsp file should have a line:

```
req.setCharacterEncoding("gb2312");
```

before any req.getParameter method calls.

This problem affects servlets and jsp files ported from earlier versions of WebSphere Application Server, which converted characters automatically based upon the locale of the WebSphere Application Server.

Scroll bars do not appear around items in the browser window

In some browsers, tree or list type items that extend beyond their allotted windows do not have scroll bars to permit viewing of the entire list.

To correct this problem, right-click on the browser window and click **Reload** from the menu.

Error "Page cannot be displayed... server not found or DNS error" appears when attempting to browse a JavaServer Pages (JSP) file using Internet Explorer

This error can occur when an HTTP timeout causes the servant to be brought down and restarted. To correct this problem, increase the ConnectionIOTimeout value:

1. From the administrative console, select **System administration > Deployment manager > Administration Services > Custom Properties**
2. Select ConnectionIOTimeout.
3. Increase the ConnectionIOTimeout value.
4. Click **OK**.

Application startup problems

When an application is not starting or starting with errors, the problem could be from one of various sources.

What kind of error do you see when you start an application?

- A "java.lang.ClassNotFoundException: classname Bean_AdderServiceHome_04f0e027Bean" on page 492 error occurs
- A "ConnectionFactory E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter" on page 492 error occurs

- “NMSV0605E: “A Reference object looked up from the context...” error when starting an application” on page 493.
- “A Page Not Found, Array Index Out of Bounds, or other error when an updated application restarts” on page 493

If none of these errors match the error you see:

- Browse the log files of the application server for this application looking for clues. By default, these files are: `app_server_root/logs/server_name/SystemErr.log` and `SystemOut.log`.
- Look up any error or warning messages in the message reference table by clicking the **Reference** view and expanding **Messages**.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting help from IBM.

java.lang.ClassNotFoundException: *classname* Bean_AdderServiceHome_04f0e027Bean

An similar exception occurs when you try to start an undeployed application containing enterprise beans, or containing undeployed enterprise bean modules.

Enterprise JavaBeans modules created in an assembly tool intentionally have incomplete configuration information. Deploying these modules completes the configuration by reading the module’s deployment descriptor and completing platform- or installation-dependent settings and adding related classes to the Enterprise JavaBeans JAR file.

To avoid this problem, do the following:

- Use an assembly tool and administrative console to generate deployment code and install the application or Enterprise JavaBeans module onto a server.
 1. Uninstall the application or Enterprise JavaBeans module in the administrative console.
 2. Configure your assembly tool so the target server is a WebSphere Application Server installation such as **WebSphere Application Server v6**. If you do not have access to the target server, you can specify a false location such as `c:\temp`. Specifying a false location enables you to assemble and generate deployment code for the enterprise bean.
 3. In the Project Explorer view of an assembly tool, right-click the enterprise bean (Enterprise JavaBeans) in the undeployed .ear file containing the Enterprise JavaBeans module or the standalone undeployed Enterprise JavaBeans JAR file, and click **Deploy**. If your assembly tool can access the WebSphere Application Server target server, deployment code is generated for the Enterprise JavaBeans and the assembly tool attempts to install the application or module onto the target server. If your assembly tool cannot access the WebSphere Application Server target server or the installation fails, use the deployment code that is generated for the next step.
For information on using an assembly tool, refer to *Assembling applications*.
 4. Use the `wsadmin $AdminApp install` command or the administrative console to install the deployed version created by the assembly tool.
- If you use the `wsadmin $AdminApp install` command, uninstall it and then reinstall using the `-EJBDeploy` option. Follow the install command with the `$AdminConfig save` command.

ConnectionFactory E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter

This error occurs when an enterprise bean developed to the Enterprise JavaBeans 1.1 specification is deployed with a WebSphere Application Server V5 J2C-compliant data source, which is the default data source. By default, persistent enterprise beans created under WebSphere Application Server V4.0 using the Application Assembly Tool fulfill the Enterprise JavaBeans 1.1 specification. To run on WebSphere Application Server V6, these enterprise beans must be associated with a WebSphere Application Server V4.0-type data source.

Either modify the mapping in the application of enterprise beans to associate 1.x container managed persistence (CMP) beans to associate them with a V4.0 data source or delete the existing data source and create a V4.0 data source with the same name.

To modify the mapping in the application of enterprise beans, in the WebSphere Application Server administrative console, select the properties for the problem application and use **map resource references to resources** or **Map data sources for all 1.x CMP beans** to switch the data source the enterprise bean uses. Save the configuration and restart the application.

To delete the existing data source and create a V4.0 data source with the same name:

1. In the administrative console, click **Resources > Manage JDBC Providers > JDBC_provider_name > Data sources**.
2. Delete the data source associated with the Enterprise JavaBeans 1.1 module.
3. Click **Resources > Manage JDBC Providers > JDBC_provider_name > Data sources (Version 4)**.
4. Create the data source for the Enterprise JavaBeans 1.1 module.
5. Save the configuration and restart the application.

NMSV0605E: "A Reference object looked up from the context..." error when starting an application

If the full text of the error is similar to:

```
[7/17/02 15:20:52:093 CDT] 5ae5a5e2 Ur1ContextHel W NMSV0605E:
A Reference object looked up from the context
"java:" with the name "comp/PM/WebSphereCMPConnectionFactory" was sent to the JNDI Naming Manager
and an exception resulted. Reference data follows:
Reference Factory Class Name: com.ibm.ws.naming.util.IndirectJndiLookupObjectFactory
Reference Factory Class Location URLs:
Reference Class Name: java.lang.Object
Type: JndiLookupInfo
Content: JndiLookupInfo: ; jndiName="eis/jdbc/MyDatasource_CMP"; providerURL="";
initialContextFactory=""
```

then the problem might be that the data source intended to support a CMP enterprise bean is not correctly associated with the enterprise bean.

To resolve this problem:

1. Select the **Use this Data Source in container managed persistence (CMP)** check box in the data source "General Properties" panel of the administrative console.
2. Verify the JNDI name in either of the following ways:
 - Verify that the JNDI name given in the administrative console under **Resources > Manage JDBC Provider > DataSource > JNDI Name** for DataSource matches the JNDI name given for CMP or BMP resource bindings at the time of assembling the application in an assembly tool.
 - Check the JNDI name for CMP or BMP resource bindings specified in the code by J2EE application developer. Open the deployed .ear folder in an assembly tool, and look for the JNDI name for your entity beans under CMP or BMP resource bindings. Verify that the names match.

A Page Not Found, Array Index Out of Bounds, or other error when an updated application restarts

If an application is updated while it is running, WebSphere Application Server automatically stops the application or only its changed components, updates the application logic, and restarts the stopped application or its components. For more information on the restarting of updated applications, refer to Fine-grained recycle behavior in *IBM WebSphere Developer Technical Journal: System management for WebSphere Application Server V6 -- Part 5 Flexible options for updating deployed applications*.

A Page Not Found, Array Index Out of Bounds, or other error might occur during restarting.

To minimize the occurrence of such errors, update applications in a test environment before updating the applications in a production environment. Do not put changes directly into a production environment.

Web resource is not displayed

Use this information to troubleshoot problems that occur when attempting to display a resource in a browser.

If you are not able to display a resource in your browser, follow these steps:

1. Verify that your HTTP server is healthy by accessing the URL `http://server_name` from a browser and seeing whether the Welcome page appears. This action indicates whether the HTTP server is up and running, regardless of the state of WebSphere Application Server.
2. If the HTTP server Welcome page does not appear, that is, if you get a browser message like page cannot be displayed or something similar, try to diagnose your Web server problem.
3. If the HTTP server appears to function correctly, the Application Server might not be serving the target resource. Try to access the resource directly through the Application Server instead of through the HTTP server.

If you cannot access the resource directly through the Application Server, verify that the URL used to access the resource is correct.

If the URL is incorrect and it is created as a link from another JavaServer Pages (JSP) file, servlet, or HTML file, try correcting it in the browser URL field and reloading, to confirm that the problem is a malformed URL. Correct the URL in the "from" HTML file, servlet or JSP file.

If the URL appears to be correct, but you cannot access the resource directly through the Application Server, verify the health of the hosting Application Server and Web module:

- a. View the hosting Application Server and Web module in the administrative console to verify that they are up and running.
- b. Copy a simple HTML or JSP file, such as `SimpleJsp.jsp`, which is in the WebSphere Application Server directory structure, to your Web module document root, and try to access the file. If successful, the problem is with the resource.

View the JVM log of your Application Server to find out why your resource cannot be found or served .

4. If you can access the resource directly through the Application Server, but not through an HTTP server, the problem lies with the HTTP plug-in, the component that communicates between the HTTP server and the WebSphere Application Server.
5. If the JSP file and the servlet output are served, but not static resources such as `.html` and image files, see the steps for enabling file serving.
6. If certain resources display correctly, but you cannot display a servlet by its class name:
 - Verify that the servlet is in a directory in the Web module class path, such as in the `/Web_module_name.war/WEB-INF/classes` directory.
 - Verify that you specify the full class name of the servlet, including its package name, in the URL.
 - Verify that `"/servlet"` precedes the class name in the URL. For example, if the root context of a Web module is "myapp", and the servlet is `com.mycom.welcomeServlet`, then the URL reads:
`http://hostname/myapp/servlet/com.mycom.welcomeServlet`
 - Verify that serving the servlets by class name is enabled for the hosting Web module by opening the source Web module in an assembly tool and browsing the *serve servlets by classname* setting in the IBM Extensions property page. If necessary, enable this flag and redeploy the Web module. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
 - For servlets or other resources served by mapped URLs, the URL is `http://hostname/Web module context root/mappedURL`.

If none of these steps fixes your problem, see if the problem has been identified and documented by looking at available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, see Troubleshooting help from IBM.

Diagnosing Web server problems

If you are unable to view the welcome page of your HTTP server, determine if the server is operating properly.

If the HTTP server does not start:

- Examine the HTTP server error log for clues.
- Try restoring the HTTP server to its configuration prior to installing WebSphere Application Server and restarting it. If you are using IBM HTTP Server:
 - Rename the file *IHS_install_dir*\httpd.conf.
 - Copy the httpd.conf.default file to the httpd.conf directory.
 - If Apache is running, stop and restart it.
- For the Sun ONE (iPlanet) Web server, restore the obj.conf configuration file for Sun ONE V4.1 and both obj.conf and magnus.conf files for Sun ONE V6.0 and later.
- For the Microsoft Internet Information Server (IIS), remove the WebSphere Application Server plug-in through the IIS administrative GUI.

If restoring the HTTP server default configuration file works, manually review the configuration file that has WebSphere Application Server updates to verify directory and file names for WebSphere Application Server files. If you cannot manually correct the configuration, you can uninstall and reinstall WebSphere Application Server to create a clean HTTP configuration file.

If restoring the default configuration file does not help, contact technical support for the Web server you are using. If you are using IBM HTTP Server with WebSphere Application Server, check available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, see Troubleshooting help from IBM

Accessing a Web resource through the application server and bypassing the HTTP server

You can bypass the HTTP server and access a Web resource through the application server. It is not recommended to serve a production Web site in this way, but it provides a good diagnostic tool when it is not clear whether a problem resides in the HTTP server, WebSphere Application Server, or the HTTP plug-in.

To access a Web resource through the Application Server:

1. Determine the port of the HTTP service in the target application server.
 - a. In the WebSphere administrative console, click **Servers > Application Servers**.
 - b. Select the target server, then under Additional Properties click **Web Container**.
 - c. Under the Additional Properties of the Web container, click **HTTP Transports**. You see the ports listed for virtual hosts served by the application server.
 - d. There can be more than one port listed. In the default application server (server1), for example, 9060 is the port reserved for administrative requests, 9443 and 9043 are used for SSL-encrypted requests. To test the sample "snoop" servlet, for example, use the default application port 9080, unless it changes.
2. Use the HTTP transport port number of the application server to access the resource from a browser. For example, if the port is 9080, the URL is `http://hostname:9080/myAppContext/myJSP.jsp`.
3. If you are still unable to access the resource, verify that the HTTP transport port is in the "Host Alias" list:
 - a. Click **Application Servers > Your_ApplicationServer > Web Container > HTTP Transports** to check the Default virtual host and the HTTP transport ports used by this application server.
 - b. Click **Environment > Virtual Hosts > default host > Host Aliases** to check if the HTTP transport port exists. Add an entry if necessary. For example, if the HTTP port for your application is server is 9080, add a host alias of `*:9082`.

Application uninstallation problems

When you try to uninstall an application or node, you might encounter problems. This topic suggests ways to resolve uninstallation problems.

What kind of problem are you having?

- After uninstalling an application through **wsadmin** tool, the application continues to run and throws "DocumentIOException"

If none of these steps fixes your problem:

- Make sure that the application and its Web and EJB modules are in a stopped state before uninstalling.
- If you are uninstalling or installing an application using **wsadmin**, make sure that you are using the **-conntype NONE** option to invoke **wsadmin** and enable local mode. To use the **-conntype NONE** option, stop the hosting application server before uninstalling the application.
- Check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).
- If you don't find your problem listed there contact IBM support

After uninstalling application through the wsadmin tool, the application throws "DocumentIOException"

If this exception occurs after the application was uninstalled using **wsadmin** with the **-conntype NONE** option:

- Restart the server or,
- Rerun the uninstall command without the **-conntype NONE** option.

Chapter 12. Troubleshooting administration

Use this information if you are having problems with administrative functions.

- Select the problem you are experiencing.
 - I have problems bringing up or using the administrative console.
 - I have problems starting or using the **wsadmin** command prompt.
 - My Web module or application server dies or hangs.
 - I get errors trying to configure and enable security.
 - I have problems creating or using HTTP sessions.
 - I have problems using tracing, logging, log files, or other troubleshooting features.
 - I get errors connecting to the administrative console from a browser.
 - I have problems using command line tools.
 - I cannot uninstall or remove a node or application server.
 - The stopServer.sh (base and ND) hangs and creates a Java core dump (Red Hat Linux).
- If you did not solve the problem, prepare to contact IBM support.

For current information available from IBM Support on known problems and their resolution, see the [IBM Support page](#).

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Administration and administrative console troubleshooting

In WebSphere Application Server products, administrative functions are supported by:

- The application server (such as server1) process

The process must be running to use the administrative console. The **wsadmin** command-line utility has a local mode that you can use to perform some administrative functions, even when the server process is not running.

What kind of problem are you seeing?

- “Server status and messages in the console view are not current”
- “Role-based authorization fails” on page 498
- “When starting or stopping a server using a wsadmin interactive scripting session, you receive an exception indicating read timed out” on page 499
- “Problems starting or using the administrative console or wsadmin utility” on page 499

Server status and messages in the console view are not current

When connecting to an Application Server that uses a Simple Object Access Protocol (SOAP) connection for a long time, the following problems begin to occur:

- Under the status column in the Servers view on an administrative console panel, the status of the server does not refresh.
- Server messages are not updated in the administrative console.
- A decrease of system resources occurs as numerous ports are created and left in the TIME_WAIT state.

This problem persists even after you restart the server or you start another server that uses the SOAP connection port.

The problem occurs because the SOAP connector does not support connection pooling. If the Application Server has many ongoing operations that use the SOAP connector, the Application Server quickly opens and closes many ports. Due to the nature of the underlying TCP/IP protocol, these ports remain in the TIME_WAIT state for some time before the operating system can reclaim them. The number of ports that

WebSphere Application Server opens can exceed the limit that the operating system imposes. Under this condition, the opening of additional ports fails through the SOAP connector until the operating system reclaims ports.

Use the following options to work around the problem:

- Increase the operating system limits on the number of ports.
- For Rational Application Developer, the wsadmin utility, or Java applications that use the Java Management Extension (JMX) connectors, switch to the Remote Method Invocation (RMI) connector.
- Wait until few or no ports are in the TIME_WAIT state before performing new operations through Rational Application Developer or the administrative console.

Role-based authorization fails

When you make a Java Management Extension (JMX) call such as `getAttribute`, `setAttribute`, `invoke`, and so on in your application, the caller requires an administrative role with sufficient permissions. The required role depends on the MBean attribute or method that the JMX caller calls and can be one of administrator, configurator, monitor, or operator. If one of the administrative roles is not assigned to the caller, or if the role is assigned, but the caller does not have the required permissions, the application receives a role-based authorization failure, for example:

```
SECJ0305I: Role based authorization check failed for securityname server.domain.name:3890/user.id,
accessId user:server.domain.name:3890/uid=user.id,ou=xxxx,dc=yyy,dc=zzz while invoking method
getNodeName on resource Server and module Server.
```

If the caller of the application cannot be assigned one of the administrative roles, the application can log in with one of the roles on behalf of the caller. For example:

```
try
{
    // Create a LoginContext to authenticate a user ID and password.
    javax.security.auth.login.LoginContext
    lc = new javax.security.auth.login.LoginContext("WSLogin",
    new com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl("adminuser",
    "adminpassword"));

    // perform the login
    lc.login();

    // Get the authenticated subject.
    javax.security.auth.Subject adminSubject = lc.getSubject();

    // Define the action that will take place using the authenticated Subject
    // You can define this action anywhere in the code, the action
    // is reference in the WSSubject.doAs that follows.
    java.security.PrivilegedAction adminAction = new java.security.PrivilegedAction()
    {
        public Object run()
        {
            try
            {
                // Get the WebSphere AdminService.
                AdminService adminservice = AdminServiceFactory.getAdminService();

                // Get the WebSphere Admin Local Server MBean instance.
                ObjectName objectname = adminservice.getLocalServer();

                // Get the Node name.
                String nodeName = (String)adminservice.getAttribute(objectname, "nodeName");

                // Get the Application Server name.
                String serverName = (String)adminservice.getAttribute(objectname, "name");

                // Get the Application Server Process ID.
```

```

String serverPid = (String)adminservice.getAttribute(objectname, "pid");

// Return a result, for this example, just return the serverPid.
return serverPid;
}
catch (Exception e)
{
e.printStackTrace();
}
return null;
}
});

// Invoke an AdminClient resource using the authenticated subject.
// This example demonstrates the action of creating an
// administrative client and returning a String value to use outside
// the doAs block.
String myData = (String)
com.ibm.websphere.security.auth.WSSubject.doAs(adminSubject, adminAction);

// use "myData" later on...
}
catch (javax.security.auth.login.LoginException e)
{
e.printStackTrace();
}
}

```

When starting or stopping a server using a wsadmin interactive scripting session, you receive an exception indicating read timed out

When starting or stopping a server using a wsadmin interactive scripting session, you receive an exception indicating read timed out, for example:

```

WASX7015E: Exception running command: "$AdminControl startServer server1 Node1";
exception information: com.ibm.websphere.management.exception.ConnectorException
org.apache.soap.SOAPException: [SOAPException: faultCode=SOAP-ENV:Client; msg=Read
timed out; targetException=java.net.SocketTimeoutException: Read timed out]

```

This exception occurs because the timeout value is too small. To fix this, increase the timeout value specified by the `com.ibm.SOAP.requestTimeout` property in the `soap.client.props` file in the `profile_root/properties` directory for a single server edition. The value you should choose depends on a number of factors such as the size and the number of the applications installed on the server, the speed of your machine, and the level of usage of your machine. The default value of the `com.ibm.SOAP.requestTimeout` property is 180 seconds.

Problems starting or using the administrative console or wsadmin utility

If you have problems starting or using the administrative console or wsadmin utility, verify that the supporting server process is started and that it is healthy.

- For the application server process, look at these files:
 - `profile_root/logs/server_name/startServer.log` for the message that indicates that the server started successfully: `ADMU3000I: Server server1 open for e-business; process id is nnnn..`
 - `profile_root/logs/server_name/SystemOut.log`
- Look up any error messages in these files in the message reference table. Select the **Reference** view in the information center navigation, and click **Messages**. A message like `WASX7213I: This scripting client is not connected to a server process when trying to start wsadmin` indicates that either the server process is not running, the host machine where it is running is not accessible, or that the port or server name that the wsadmin utility uses is incorrect.
- Verify that you are using the right port number to communicate with the administrative console or the wsadmin server:
 - Look in the `SystemOut.log` file.

- The line `ADMC0013I: SOAP connector available at port nnnn` indicates the port that the server is using to listen for `wsadmin` functions.
- The `com.ibm.ws.scripting.port` property in the `profile_root/properties/wsadmin.properties` file controls the port used by the `wsadmin` utility to send requests to the server.
- If the port value is different from the value shown in the `SystemOut.log` file, either change the port number in the `wsadmin.properties` file, or specify the correct port number when starting the `wsadmin` utility by using the `-port port_number` property on the command line.

The `com.ibm.ws.scripting.port` property in the `profile_root/properties/wsadmin.properties` file controls the port used by the `wsadmin` utility to send requests to the server.
- If the port value is different from the value shown in the server log files, either change the port number in the `wsadmin.properties` file, or specify the correct port number when starting the `wsadmin` utility by using the `-port port_number` property on the command line.

The message `SRVE0171I: Transport http is listening on port nnnn` (default 9060) indicates the port that the server uses to listen for administrative console requests.
- If the port value is different than the one specified in the Web address for the administrative console, change the Web address in the browser to the correct value. The default value is `http://localhost:9060/ibm/console`.
- Use the **telnet** command to test that the host name where the application server is running, is reachable from the system where the browser or `wsadmin` program is used. If you can ping the host name, no firewall or connectivity issues exist.
- If the host where the application server is running is remote to the machine from which the client browser or `wsadmin` command is running, ensure that the appropriate host name parameter is correct. Verify:
 - The host name in the browser Web address for the console.
 - The `-host host name` option of the **wsadmin** command that is used to direct the `wsadmin` utility to the right server
- Tracing the administrative component: WebSphere Application Server technical support might ask you to trace the administrative component for detailed problem determination. The trace specification for this component is `com.ibm.websphere.management.*=all=enabled:com.ibm.ws.management.*=all=enabled"`

If none of these steps solves the problem, see if the specific problem you are having is addressed in the Installation completes but the administrative console does not start topic. Check to see if the problem is identified and documented using the links in the Diagnosing and fixing problems: Resources for learning topic.

For current information available from IBM Support on known problems and their resolution, see the following topics on the IBM support page:

- Administrative Console
- Administrative Scripting Tools
- System management

IBM Support has documents that can save you time gathering the information that is needed to resolve this problem. Before opening a PMR, see the following topics on information gathering on the IBM support page:

- Administrative Console
- Administrative Scripting Tools
- System management

Administrative console does not start even though installation completes

This topic discusses problems that you can encounter when you attempt to access the console.

What kind of problem are you having?

- “An Internal Server Error, Page cannot be found, 404, or similar error occurs trying to view the administrative console”
- “An Unable to process login. Check user ID and password and try again. error occurs when trying to access the administrative console page”
- “The directory paths in the administrative console contain strange characters”

If you can bring up the browser page, but the administrative console behavior is inconsistent, error prone, or unresponsive, try upgrading your browser. Older browsers might not support all the features of the administrative console.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in Troubleshooting help from IBM. Before opening a problem report, see the Support page:

- <http://www.ibm.com/servers/eserver/support/series/allproducts/index.html>

An Internal Server Error, Page cannot be found, 404, or similar error occurs trying to view the administrative console

Here are some steps to try if you are unable to view the administrative console:

- Verify that the application server that supports the administrative console is up and running. For a base configuration, the administrative console is deployed by default on server1.
Before viewing the administrative console, you must run the startServer Qshell script. For more information on starting the server, see Starting an application server.
- View the SystemOut.log file for the application server to verify that the server that supports the administrative console started.
- Check the Web address you use to view the console. By default, this address is `http://server_name:9060/ibm/console`, where `server_name` is the host name.
- If you are browsing the administrative console from a remote machine, try to eliminate connection, address and firewall issues by pinging the server machine from a command prompt, using the server name in the Web address.
- If you have never been able to access the administrative console see the topic on installation troubleshooting tips.

An Unable to process login. Check user ID and password and try again. error occurs when trying to access the administrative console page

This error indicates that security is enabled for WebSphere Application Server, and that the user ID or password supplied is either not valid or not authorized to access the console.

To access the console:

- If you are the administrator, use the ID defined as the security administrative ID. This ID is stored in the WebSphere Application Server security.xml file.
- If you are not the administrator, ask the administrator to enable your ID for the administrative console.

The directory paths in the administrative console contain strange characters

Directory paths that are used for class paths or resources specified in an assembly tool, in configuration files, or elsewhere that contain strange characters when they are viewed in the administrative console might result from the Java run time interpreting a backslash (\) as a control character.

To resolve this problem, modify Windows-style class paths by replacing occurrences of single back slashes to two. For example, change `c:\MyFiles\MyJsp.jsp` to `c:\\MyFiles\\MyJsp.jsp`.

Administrative console - browser connection problems

This topic describes problems that you can have when logging into the administrative console from a browser.

Review the following information to resolve your browser problem.

If you are able to bring up the browser page, but the console behavior is inconsistent, error-prone, or unresponsive, try upgrading the browser you are using. Older browsers may not support the administrative console's features. For a listing of supported Web browsers, see the Supported hardware and software Web page.

Check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Check the following list for your problem and how to solve it:

- When a single user that uses multiple instances of the Mozilla browser logs into the administrative console, the first user ID that logs into the administrative console is the current user.
- A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.
- A user on Mozilla browser Version 1.4 enters an invalid ID or password, presses Enter, and receives an error message

When a single user that uses multiple instances of the Mozilla browser logs into the administrative console, the first user ID that logs into the administrative console is the current user.

When a single user logged into an operating system tries to use multiple instances of the Mozilla browser, the first user ID that logs into the administrative console is the current user. This situation occurs because the browser windows share a single process.

To resolve the problem, do one of the following actions:

- Have single users logged into an operating system use a single instance of the Mozilla browser to log into the administrative console.
- If the operating system allows multiple users on an operating system, have each user log into the operating system with a different user ID and bring up a single instance of the Mozilla browser.

A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.

A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.

To resolve the problem, do one of the following actions:

- Explicitly select a button of interest on the administrative console panel instead of pressing Enter.
- Use a later version of a supported Mozilla browser.
- Use a supported version of the Microsoft Internet Explorer browser.

A user on Mozilla browser Version 1.4 enters an invalid ID or password, presses Enter, and receives an error message

A user on Mozilla browser Version 1.4 enters an invalid user ID or password, presses Enter, and receives an error message. Clicking OK fails to refresh the administrative login screen

To resolve the problem, do one of the following actions:

- Use a later version of a supported Mozilla browser.
- Use a supported version of the Microsoft Internet Explorer browser.

Web server plug-in troubleshooting tips

The following topics might help you diagnose problems with the Web server plug-ins.

If you are having problems using a Web server plug-in, try these steps:

- Review the file *plugins_root/logs/web_server_name/http_plugin.log* for clues. Look up any error or warning messages in the message table.
- Review your Web server's error and access logs to see if the Web server is having a problem:
 - IBM HTTP Server and Apache: *access.log* and *error.log*.
 - Domino Web server: *httpd-log* and *httpd-error*.
 - Sun Java System: *access* and *error*.
 - Microsoft IIS: *timedatestamp.log*.

If these files don't reveal the cause of the problem, follow these additional steps.

Plug-in Problem Determination Steps

The plug-in provides very readable tracing which can be beneficial in helping to figure out the problem. By setting the *LogLevel* attribute in the *config/plugin-cfg.xml* file to **Trace**, you can follow the request processing to see what is going wrong.

Note: If you are using a Veritas File System with large file support enabled, file sizes up to two terabytes are allowed. In this case, if you set the *LogLevel* attribute in the *plugin-cfg.xml* file to *LogLevel=Trace*, then the *http_plugin.log* file might grow quickly and consume all available space on your file system. Therefore, you should set the value of the *LogLevel* attribute to **ERROR** or **DEBUG** to prevent high CPU utilization..

At a high level, complete these steps.

1. The plug-in gets a request.
2. The plug-in checks the routes defined in the *plugin-cfg.xml* file.
3. It finds the server group.
4. It finds the server.
5. It picks the transport protocol, HTTP or HTTPS.
6. It sends the request.
7. It reads the response.
8. It writes it back to the client.

You can see this very clearly by reading through the trace for a single request:

- The first step is to determine if the plug-in has successfully loaded into the Web server.
 - Check to make sure the *http_plugin.log* file has been created.
 - If it has, look in it to see if any error messages indicate some sort of failure that took place during plug-in initialization. If no errors are found look for the following stanza, which indicates that the plug-in started normally. Ensure that the timestamps for the messages correspond to the time you started the Web server:

```
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: -----System Information-----
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: Bld date: Jul 3 2002, 15:35:09
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: Web server: IIS
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: Hostname = SWEETTJ05
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: OS version 4.0, build 1381, 'Service Pack 6'
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: -----
```

- Some common errors are:

lib_security: loadSecurityLibrary: Failed to load gsk library

Either GSKit did not get installed or the wrong version of GSKit got installed. To determine which situation occurred:

If you cannot find the appropriate file, try reinstalling the plug-in with the correct GSKit version to see if this fixes the problem.

ws_transport: transportInitializeSecurity: Keyring wasn't set

The HTTPS transport defined in the configuration file was prematurely terminated and did not contain the Property definitions for the keyring and stashfile. Check your XML syntax for the line number given in the error messages that follow this one to make sure the Transport element contains definitions for the keyring and stashfiles before it is terminated.

- If thehttp_plugin.log file is not created, check the Web server error log to see if any plug-in related error messages have been logged there that indicate why the plug-in is failing to load. Typical causes of this can include failing to correctly configure the plug-in with the Web server environment. Check the documentation for configuring the Web server that you are using with the Web server plug-in.
- Determine whether there are network connection problems with the plug-in and the various application servers defined in the configuration. Typically you will see the following message when this is the case:

ws_common: websphereGetStream: Failed to connect to app server, OS err=%d

Where %d is an OS specific error code related to why the connect() call failed. This can happen for a variety of reasons.

- Ping the machines to make sure they are properly connected to the network. If the machines cannot be pinged, there is no way for the plug-in to contact them. Possible reasons for this problem include:
 - Firewall policies that limit the traffic from the plug-in to the application server.
 - The machines are not on the same network.
- If you are able to ping the machines then the probable cause of the problem is that the port is not active. The port might not be active because the application server or cluster is not started or the application server has gone down for some reason. To verify that this is the problem, you can try to manually telnet into the port that the connect is failing on. If you cannot telnet into the port the application server is not up and that problem needs to be resolved before the plug-in can successfully connect.
- Determine whether other activity on the machines where the servers are installed is impairing the ability of the server to service a request. Check the processor utilization as measured by the task manager, processor ID, or some other outside tool to see if it:
 - Is not what was expected.
 - Is erratic rather than a constant.
 - Shows that a newly added member of the cluster is not being utilized.
 - Shows that a failing member that has been fixed is not being utilized.
- Check the administrative console for server status.

Check the administrative console to ensure that the application server is started. View the administrative console for error messages or look in the JVM logs.

- In the administrative console, select the problem application server and view its installed applications to verify that they are started.

If none of these steps solves the problem:

- For specific problems that can cause web pages and their contents not to display, see Web resource (JSP file, servlet, html file, image, etc) will not display in the information center.
- Check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning.
- If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

For current information available from IBM Support on known problems and their resolution, see the following topics on the IBM support page:

- HTTP transport
- HTTP plug-in
- HTTP plug-in remote install

For current information available from IBM Support on known problems and their resolution, see the IBM Support page. You should also refer to this page before opening a PMR because it contains documents that can save you time gathering information needed to resolve a problem.

You might find the following topics on the IBM support page helpful:

- HTTP plug-in
- HTTP plug-in remote install

Administrative scripting problems

Use this information if you are having problems starting or using the wsadmin tool.

What kind of problem are you having?

- WASX7016E, WASX7017E, or WASX7209I: Jython scripting language error
- "WASX7023E: Error creating "SOAP" connection to host" or similar error trying to launch wsadmin command line utility.
- "com.ibm.bsf.BSFException: error while evaluating Jacl expression: no such method "<command name>" in class com.ibm.ws.scripting.AdminConfigClient" returned from wsadmin command.
- WASX7022E returned from running "wsadmin -c ..." command, indicating invalid command.
- com.ibm.ws.scripting.ScriptingException: WASX7025E: String "" is malformed; cannot create ObjectName.
- "The input line is too long" error returned from the wsadmin command on a Windows platform.
- WASX701E: Exception received while running file "*scriptName.jacl*"; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: missing close-bracket
- WASX7015E: Exception running command: "source c: ..."; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: couldn't read file "c: ..."
- Unexpected error CWSIV0806E in WebSphere log following deletion of an outbound service
- Separator exception
- The format of "\$AdminConfig list" output has changed for V6.0
- You are not prompted for user ID and password after applying V6.0.2 service if you use an existing 6.0 profile
- Required ports not defined when a new V5.0/V5.1 server is created
- When running the \$AdminApp searchJNDIReferences command with the Java Naming and Directory Interface (JNDI) name of a message destination, the message destination reference is not returned

If you do not see your problem here:

- If you are not able to enter wsadmin command mode, try running **wsadmin -c "\$Help wsadmin"** for help in verifying that you are entering the command correctly.
- If you can get the wsadmin command prompt, enter **\$Help help** to verify that you are using specific commands correctly.
- wsadmin commands are a superset of Jacl (Java Command Language), which is in turn a Java-based implementation of the Tcl command language. For details on Jacl syntax beyond wsadmin commands, refer to the Tcl developers' site, <http://www.tcl.tk>. For specific details relating to the Java implementation of Tcl, refer to <http://www.tcl.tk/software/java>.
- Browse the *install_dir/profiles/profile_name/logs/wsadmin.traceout* file for clues.
 - Keep in mind that wsadmin.traceout is refreshed (existing log records are deleted) whenever a new wsadmin session is started.
 - If the error returned by wsadmin does not seem to apply to the command you entered, for example, you receive WASX7023E, stating that a connection could not be created to host "myhost," but you

did not specify "-host myhost" on the command line, examine the properties files used by wsadmin to determine what properties are specified. If you do not know what properties files were loaded, look for the WASX7326I messages in the wsadmin.traceout file; there will be one of these messages for each properties file loaded.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there please contact IBM support.

WASX7016E, WASX7017E, or WASX7209I: Jython scripting language error

The following errors may occur when you run this Jython script:

Jython script

```
"profile_root/bin/wsadmin.sh -lang jython -profile profile_name -host host_name -f
script_file.py"
```

Error Messages

```
WASX7209I: Connected to process "server1" on node
node_name using SOAP connector; The type of process is:
UnManagedProcess
```

```
WASX7016E: Exception received while reading file
"script_file.py"; exception information:
sun.io.MalformedInputException
```

```
WASX7017E: Exception received while running file
"script_file.py"; exception information:
com.ibm.bsf.BSFException: exception from Jython: Traceback
(innermost last): File "<string>" line 89, in ? NameError: log
```

These errors can occur because there are UTF-8 characters in the file that are not valid. The default codepage for RHEL 3 is UTF-8 (en_US.UTF-8). When doing a text file read through Java™ code, the program assumes all characters are UTF-8 encoded. There might be one or more characters in the file that are not part of the UTF-8 specification, causing the load to fail. An easy way to determine if a character that is not valid is causing the error is to enter export LANG=C and run the script again. If you determine that the problem is a character that is not valid:

1. Open a new text reader on the file.
2. Read it one character at a time.
3. Print the character that is not valid.
4. When you press the back characters, you get the exception and will then know which of the characters is causing the error.
5. Remove any characters that are not valid, then run the script again

"WASX7023E: Error creating "SOAP" connection to host" or similar error trying to launch wsadmin command line utility

By default, the wsadmin utility attempts to connect to an application server at startup. This is because some commands act upon running application servers. This error indicates that no connection could be established.

To resolve this problem:

- If you are not sure whether an application server is running, start it by entering **startserver servername** from the command prompt. If the server is already running, you will see an error similar to "ADMU3027E: An instance of the server is already running".
- If an application server is running and you still get this error:

- If you are running remotely (that is, on a different machine from the one running WebSphere Application Server), you must use the **-host *hostname*** option to the wsadmin command to direct wsadmin to the right physical server.
- If you are using the -host option, try pinging the server machine from the command line from the machine on which you are trying to launch wsadmin to verify there are no issues of connectivity such as firewalls.
- verify that you are using the right port number to connect to the WebSphere Application Server process:
 - If you are not specifying a port number (using the -port option) when you start the wsadmin tool, the wsadmin tool uses the default port specified in *install_dir/profiles/profile_name/properties/wsadmin.properties* file, property name=`com.ibm.ws.scripting.port` (default value =8879).
 - The port that wsadmin should send on depends on the server process wsadmin is trying to connect to.

For a single-server installation, wsadmin attempts to connect to the application server process by default. To verify the port number:

- Look in the file *profile_root/config/cells/cell_name/nodes/node_name/serverindex.xml* for a tag containing the property **serverType="APPLICATION_SERVER"**.
- Look for an entry within that tag with the property **endpointName="SOAP_CONNECTOR_ADDRESS"**.
- Look for a **port** property within that tag. This is the port wsadmin should send on.
- If the keyring settings seem correct, enable tracing to display more detailed information about the error. Start the scripting client with the following debug options:

```
wsadmin.sh -javaoption "-Djava.security.auth.debug=all -Djavax.net.debug=true"
```

"com.ibm.bsf.BSFException: error while eval'ing Jacl expression: no such method *command name* in class com.ibm.ws.scripting.AdminConfigClient" returned from wsadmin command.

This error is usually caused by a misspelled command name. Use the **\$AdminConfig help** command to get information about what commands are available. Note that command names are case-sensitive.

WASX7022E returned from running "wsadmin -c ..." command, indicating invalid command

If the command following -c appears to be valid, the problem may be caused by the fact that on Unix, using wsadmin -c to invoke a command that includes dollar signs results in the shell attempting to do variable substitution. To confirm that this is the problem, check the command to see if it contains an unescaped dollar sign, for example: **wsadmin -c "\$AdminApp install"**.

To correct this problem, escape the dollar sign with a backslash. For example: **wsadmin -c "\\$AdminApp install ..."**.

com.ibm.ws.scripting.ScriptingException: WASX7025E: String "" is malformed; cannot create ObjectName

One possible cause of this error is that an empty string was specified for an object name. This can happen if you use one scripting statement to create an object name and the next statement to use that name, perhaps in an "invoke" or "getAttribute" command, but you don't check to see if the first statement really returned an object name. For example (the following samples use basic Jacl commands in addition to the wsadmin Jacl extensions to make a sample script):

```
#let's misspell "Server"
set serverName [$AdminControl queryNames type=Srever,*]
$AdminControl getAttributes $serverName
```

To correct this error, make sure that object name strings have values before using them. For example:

```

set serverName[${AdminControl queryNames node=mynode,type=Server,name=server1,*}]
if {$serverName == ""} {puts "queryNames returned empty - check query argument"}
else {$AdminControl getAttributes $serverName}

```

For details on Jacl syntax beyond wsadmin commands, refer to the Tcl developers' site, <http://www.tcl.tk>.

"The input line is too long" error returned from the wsadmin command on a Windows platform

This error indicates that the Windows command line limit of 2048 characters has been exceeded, probably due to a long profile path used within the **wsadmin.bat** command. You may get this error when running wsadmin in a Windows command prompt or calling wsadmin from a .bat file, an ant build file, or Profile Management Tool. If this error results in running wsadmin other than from the Profile Management Tool, avoid the problem by using the Windows **subst** command, which allows you to map an entire path to a virtual drive. To see the syntax of the **subst** command, enter `help subst` from a Windows command prompt.

For example, if the product resides in the *app_server_root* directory, edit the *app_server_root\bin\setupCmdLine.bat* file as follows:

```

SET CUR_DIR=%cd%
cd /d "%~dp0.."
SET WAS_HOME=%cd%
cd /d "%CUR_DIR%"

```

```

@REM add the following two lines to workaround Windows 2K command line length limit
subst w: %WAS_HOME%
set WAS_HOME=w:

```

```

...
...

```

Then edit the *setupCmdLine.bat* file residing in the bin directory of your profile as follows:

```

SET WAS_USER_PROFILE=...
SET USER_INSTALL_ROOT=...
SET WAS_HOME=app_server_root
SET JAVA_HOME=app_server_root\java

```

```

@REM add the following three lines to workaround Windows 2K command line length limit
subst w: %WAS_HOME%
set WAS_HOME=w:
set JAVA_HOME=%WAS_HOME%\java

```

```

...
...

```

If this error occurred while running the Profile Management Tool, you have to rerun the Profile Management Tool to provide a shorter profile path with a shorter profile name. If this does not fix the problem, follow the same instructions above to edit the *setupCmdLine.bat* file in the bin directory of your WebSphere Application Server installation. After editing the file, rerun the Profile Management Tool. If the same problem persists, reinstall WebSphere Application Server with a shorter installation root directory path.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

WASX701E: Exception received while running file "*scriptName.jacl*"; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: missing close-bracket

This error is caused by a mix-up between the code page that the scripting client expects to see and the code page in which the Jacl script was written.

To fix this problem, set the `-Dscript.encoding=script codepage` option in the `wsadmin.sh` or `wsadmin.bat` file to the code page of the Jacl script. The following guideline will help you to determine the code page of the script:

- If the script was written in the OMVS interface using the OEDIT editor, the code page is IBM-037. In this case, set the option to the following: `-Dscript.encoding=Cp037`
- If the script was written in a telnet session to the OMVS interface using the VI editor, the code page is IBM-1047. In this case, set the option to the following: `-Dscript.encoding=Cp1047`
- IF the script was written on a personal computer, or any other ASCII machine, and was transferred to the host as a text file, the code page is IBM-1047. In this case, set the option to the following: `-Dscript.encoding=Cp1047`
- If the script was written on a personal computer, or any other ASCII machine, and transferred to the host in binary format, the code page is ISO-8859-1 (ASCII). In this case, you do not need to set the option because the default is ASCII. You should review other possible reasons for this error.

WASX7015E: Exception running command: "*source c: ...*"; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: couldn't read file "*c: ...*"

This error is caused by using a backslash (\) instead of a forward slash (/) when running the `wsadmin` command to source a Jacl script in a Windows® environment. The file path cannot contain the backslash (\); for example, `wsadmin> source c:\temp\test.jacl`. The file path must use the forward slash (/) as the path separator; for example, `wsadmin> source c:/temp/test.jacl`.

To correct this problem use the forward slash (/) in the file path when using the `wsadmin` command to source a Jacl script in a Windows® environment:

```
app_server_root\bin>wsadmin
WASX7209I: Connected to process "dmgr" on node sunCellManager01
using SOAP connector; The type of process is:
DeploymentManager WASX7029I: For help, enter: "$Help help"
wsadmin>source c:/temp/test.jacl
```

Unexpected error CWSIV0806E in WebSphere log following deletion of an outbound service

This error occurs when an exception is issued for destination `MPOutBoundServicePortDestination`, on messaging engine `trueliesNode01.server1-FVTSIBus01`, on bus `FVTSIBus01`, for endpoint activation:

```
com.ibm.websphere.sib.exception.SINotPossibleInCurrentConfigurationException: CWSIP0111E: The destination with name MPOutBoundServicePortDestination is being deleted on messaging engine {1}.
```

You can ignore this error; it is benign.

Separator exception

You must use forward slashes (/) as your path separator. Backward slashes (\) will not work.

The format of "\$AdminConfig list" output has changed for V6.0

If you have a script that parses the output of \$AdminConfig list, such as \$AdminConfig list Node, you might receive errors, such as "Node not found." Scripts should not parse the output of \$AdminConfig; however, if you have a script that does this parsing, it must be updated for WebSphere Application Server V6.0 to reflect changes to the output format.

You are not prompted for user ID and password after applying V6.0.2 service if you use an existing 6.0 profile

If security is enabled, executing a .bat file requires a user ID and password. On V6.0.2, a new feature is introduced to prompt you for a user ID and password if they are not supplied in the command line. However, this feature is not available for profiles that were created at the 6.0 level.

Property files for profiles created at the V6.0 level are not updated after applying the V6.0.2 refresh pack.

There are two solutions to this problem:

1. Create a new profile after applying the V6.0.2 service. This new profile contains all the updated property files and you will then be prompted for a user ID and password.
2. If you want to keep the existing V6.0 profile and use the new prompt feature, you must manually update three files:
 - for `app_server_root/properties/soap.client.props`, add the following line:
`com.ibm.SOAP.loginSource=prompt`
 - for `app_server_root/properties/wsjaas_client.conf`, add the following lines:

```
WSAdminClientLogin {
  com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy required del
  egate=com.ibm.ws.security.common.auth.module.WSAdminClientLoginModuleImpl;
};
```
 - for `app_server_root/bin/setupCmdLine.bat` add the following line:
`SET JAASSOAP=-Djava.security.auth.login.config=app_server_root/properties/wsjaas_client.conf`

Required ports not defined when a new V5.0/V5.1 server is created

Required ports not defined when a new V5.0/V5.1 server is created

You create a new V5.0 or V5.1 server using a default template in a mixed cell environment, and all of the required ports are not defined for the newly created server.

Add the following two entries to the `serverindex.xml` file in the `profile_root/config/templates/servertypes/APPLICATION_SERVER/` directory:

- ```
<serverEntries xmi:id="ServerEntry_3" serverName="default_5X"
serverType="APPLICATION_SERVER">
 <specialEndpoints xmi:id="NamedEndPoint_25" endPointName="BOOTSTRAP_ADDRESS">
 <endPoint xmi:id="EndPoint_25" host="$(node.host.name)" port="2809"/>
 </specialEndpoints>
 <specialEndpoints xmi:id="NamedEndPoint_26" endPointName="SOAP_CONNECTOR_ADDRESS">
 <endPoint xmi:id="EndPoint_26" host="$(node.host.name)" port="8880"/>
 </specialEndpoints>
 <specialEndpoints xmi:id="NamedEndPoint_27" endPointName="SAS_SSL_
SERVERAUTH_LISTENER_ADDRESS">
 <endPoint xmi:id="EndPoint_27" host="$(node.host.name)" port="0"/>
 </specialEndpoints>
 <specialEndpoints xmi:id="NamedEndPoint_28" endPointName="CSIV2_
SSL_SERVERAUTH_LISTENER_ADDRESS">
 <endPoint xmi:id="EndPoint_28" host="$(node.host.name)" port="0"/>
 </specialEndpoints>
```

```

<specialEndpoints xmi:id="NamedEndPoint_29" endPointName="CSIV2_
SSL_MUTUALAUTH_LISTENER_ADDRESS">
<endPoint xmi:id="EndPoint_29" host="$(node.host.name)" port="0"/>
</specialEndpoints>
</serverEntries>
• <serverEntries xmi:id="ServerEntry_4" serverDisplayName="default
ZOS_5X" serverName="defaultZOS_5X" serverType="APPLICATION_SERVER"
serverUniqueId="BB80B67909190083000000DC0010200209390F08">
<specialEndpoints xmi:id="NamedEndPoint_31" endPointName="BOOTSTRAP_ADDRESS">
<endPoint xmi:id="EndPoint_31" host="$(node.host.name)" port="2809"/>
</specialEndpoints>
<specialEndpoints xmi:id="NamedEndPoint_32" endPointName="SOAP_CONNECTOR_ADDRESS">
<endPoint xmi:id="EndPoint_32" host="$(node.host.name)" port="8880"/>
</specialEndpoints>
<specialEndpoints xmi:id="NamedEndPoint_33" endPointName="ORB_SSL_LISTENER_ADDRESS">
<endPoint xmi:id="EndPoint_33" host="*" port="0"/>
</specialEndpoints>
<specialEndpoints xmi:id="NamedEndPoint_34" endPointName="ORB_LISTENER_ADDRESS">
<endPoint xmi:id="EndPoint_34" host="*" port="0"/>
</specialEndpoints>
</serverEntries>

```

The xmi:ids must all be unique within this file.

## When running the \$AdminApp searchJNDIReferences command with the Java Naming and Directory Interface (JNDI) name of a message destination, the message destination reference is not returned

This problem occurs when the command \$AdmnApp searchJNDIReferences is run with the JNDI name of a message destination. The command cannot collect the message destination reference that is defined in the application deployment descriptor. The message destination that you configured for the application server is defined with a message destination link on not one element, but two: both a message-driven bean (MDB) and a message destination reference.

Currently there is no workaround for this problem. The \$AdmnApp searchJNDIReferences command cannot return a reference for a message destination that is defined on two elements.

---

## Tracing and logging facilities - troubleshooting tips

Use this information if you are having problems using tracing, logging or other troubleshooting tools.

What kind of problem are you having?

- Error messages when launching the Log Analyzer.
- Netscape browser fails when trying to enable a component trace.

### Error messages when launching the Log Analyzer

Upon starting the Log Analyzer for the first time or after the Log Analyzer preferences files of the users are deleted, the following message displays in the Log Analyzer shell window:

```
Cannot open input stream for waslogbrsys
```

This message is an informational message. You can disregard the message because it does not affect the execution of the Log Analyzer.

The following error messages might display in the Log Analyzer shell window when you start the Log Analyzer:

```
Cannot open input stream for default
```

```
Cannot open input stream for default
```

```
Cannot load configuration: default
Cannot open input stream for default
Cannot open input stream for default
Cannot load configuration: default
```

These error messages indicate corrupt or incomplete user preference files.

To resolve this problem, take the following steps:

1. Close the Log Analyzer.
2. Delete all user preference files in the `%USERPROFILE%\logbr` directory on Windows platforms or `$HOME/logbr` directory on UNIX platforms.
3. Restart the Log Analyzer.

**Note:**Deleting all user preference files removes the preferences of Log Analyzer set by the user in the preferences dialog.

## Netscape browser fails when trying to enable a component trace

On systems using AIX, the Netscape browser fails when you try to enable trace on a component.

To work around this problem, do one of the following:

- Disable JavaScript on the browser and continue setting trace.
- Administer the AIX server from a remote machine running another browser and operating system.
- Change the trace manually in the `server.xml` file.

---

## Server startup problems

If a server process does not start or starts with errors, the following topics might help you to diagnose the problem.

### Installation program completes successfully, but an application server does not start, or starts with errors

- Browse the Application Server log files for clues. The log files are located by default in:
  - `profile_root/logs/server_name/SystemErr.log` and `SystemOut.log`

Several applications deployed on an application server or node can take time to start. Browse the `SystemOut.log` periodically and look at the most recent updates to see if the server is still starting up.

- Look for any errors or warnings relating to specific resources with the module, such as Web modules, enterprise beans and messaging resources. If you find any, examine the application server configuration file for the configuration settings of that resource. Then restart the server to see if this component causes the problem.

For example, browse `profile_root/config/cells/ApplicationServerCell/nodes/node_name/servers/server_name/server.xml`, and examine the XML tags for the properties of that resource. Change its **initialState** value from `START` to `STOP`.

- Look up any error or warning messages in the message reference table by clicking the **Reference** view of the information center navigation and expanding **Messages** in the navigation tree.
- After you create an application server, you must synchronize the nodes before saving the configuration settings for the new server. If you do not synchronize the nodes, your new server might not start.
  1. On the Applications server page listing all of your application server, click **Preferences**.
  2. Select **Synchronize changes with Nodes**, if it is not already selected.
  3. Click **Apply** and then click **Application servers** to return to your list of application servers.
  4. Click **Save** to save the configuration settings for the new server.
- Verify that the logical name that you specified to appear on the console for your application server does not contain invalid characters like: `- / \ : * ? " < >` and leading or trailing spaces.
- If you are using Apache Derby and receive an `ERROR XSDB6: Another instance of Apache Derby might have already booted the database databaseName` error when starting the application server, consult the topic *Data access problems* for more information.

- When using a user profile other than QEJBSVR to run an application server, verify that:
  - The user profile has QEJBSVR specified as its group profile.
  - QEJBSVR is the owner of all files and directories under the *profile\_root* directory. You can use the following command in a Qshell session to set QEJBSVR as the owner:
 

```
chown -R QEJBSVR profile_root
```
- The application server might not start in the restricted mode. You can configure an application server to allow or restrict access to internal server classes. The default is to allow access. If access is restricted, the server might not start. If the application server does not start in *Restrict* mode, change the access to internal classes to *Allow*.

**Message "The socket bind failed for host *hostname* and port *portnumber*. The port may already be in use." occurs when restarting an application server.**

The following error message might appear in the SystemOut.log after restarting an application server:  
The socket bind failed for host *hostname* and port *portnumber*. The port may already be in use.

This problem might occur if the network is slow, and the port listed in the message text did not finish listening when the application was stopped and restarted.

To verify that this is the problem, check the port status.

To correct this problem, wait for a few minutes after stopping the server:

1. Verify that no ports are listening. Use this CL command:

```
NETSTAT *CNN
```

2. Restart the server

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in Troubleshooting help from IBM. Before opening a problem report, see the Support page:

- <http://www.ibm.com/servers/eserver/support/series/allproducts/index.html>

---

## Server hangs during shutdown if it creates a Java core dump (Red Hat Linux)

When you run the stopServer.sh script on Red Hat Linux Advanced Server Version 2.1 with the latest operating system patches, it creates a Java core dump and hangs the terminal.

To fix this problem:

- kill all Java and MQ processes
- uninstall the latest version of GNU Standard C++ Library
  - run the command `rpm -e --nodeps libstdc++-2.96-116.7.2`
- Reinstall the older version from Redhat Advanced Server V2.1 CD
  - run the command `rpm -ihv libstdc++-2.96-108.1.rpm`

---

## Command line tool problems

This article provides troubleshooting support for a variety of problems relating to using command line tools.

What kind of problem are you having?

- Just-in-time (JIT) compiler is disabled when you start application server with DEBUG enabled on a Red Hat Linux machine
- The startServer.sh or stopServer.sh commands fail to start or stop the server when the server definition is part of the configuration repository.

- The stopServer command fails to stop the server because the system cannot create a connector to an invalid hostname.

### **Just-in-time (JIT) compiler is disabled when you start the application server with DEBUG enabled on a Red Hat Linux machine**

The just-in-time (JIT) compiler is disabled when you start the application server with Software Developer Kit (SDK) DEBUG enabled on a Red Hat Linux® machine, even though JIT is set to enabled. To verify this setting, check the SystemOut.log or the startServer.log file.

Use the administrative console to remove the following DEBUG options of the Java™ process definition.  
-Xdebug -Xnoagent

For more information, see the V6.0 Information Center Release Notes page.

### **The startServer.sh or stopServer.sh commands fail to start or stop the server when the server definition is part of the configuration repository.**

This problem occurs when the startServer.sh or stopServer.sh commands are trying to start or stop non Java™ process. To solve this problem, use the -nowait option to start or stop the server, for example:

```
startServer.sh webserver1 -nowait
stopServer.sh webserver1 -nowait
```

### **With Windows service, there is no indication when a server is already started.**

When attempting to start an already-started server from the command line, there is no indication that the server is already started and running. When running startManager.bat on Windows® the following output is displayed before the command returns:

```
ADMU7701I: Because dmgr is registered to run as a Windows Service, the request to start this server will be completed by starting the associated Windows Service.
```

When running startServer.bat, the following output is displayed before the command returns:

```
ADMU7701I: Because server1 is registered to run as a Windows Service, the request to start this server will be completed by starting the associated Windows Service.
```

When running WASService.exe, the following output is displayed before the command returns:

```
Starting Service: service name
```

To check if the server is started or if the service is running, use the serverStatus *server name* command or the WASService -status *service name* command.

### **The stopServer command fails to stop the server because the system cannot create a connector to an invalid hostname.**

If the stopServer command fails to stop the server because the system cannot create a connector to an invalid hostname, you can stop the server using one of following methods:

- Stop the server process on the operating system (e.g., on Unix based systems issue the kill command).  
Or
- Open a wsadmin tool connection directly to the connector port of the server and call the stop method for the MBean of the server. This method is recommended because it allows ongoing work to shut down gracefully.

1. Issue the following command to connect to the server:

```
wsadmin -host <host_name> -port <connector_port> -conntype [SOAP | RMI] -user <user ID> -password <password>
```

2. Invoke the stop method on the MBean of the server. For example, in Jython you can use:

```
serverMBean = AdminControl.completeObjectName("*,type=Server")
AdminControl.invoke(serverMBean, "stop")
```

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there contact IBM support.





---

## Appendix. Directory conventions

References in product information to *app\_server\_root*, *profile\_root*, and other directories infer specific default directory locations. This topic describes the conventions in use for WebSphere Application Server.

### Default product locations (i5/OS)

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server products or components require multiple locations.

#### *app\_client\_root*

The default installation root directory for the Java EE WebSphere Application Client is the /QIBM/ProdData/WebSphere/AppClient/V7/client directory.

#### *app\_client\_user\_data\_root*

The default Java EE WebSphere Application Client user data root is the /QIBM/UserData/WebSphere/AppClient/V7/client directory.

#### *app\_client\_profile\_root*

The default Java EE WebSphere Application Client profile root is the /QIBM/UserData/WebSphere/AppClient/V7/client/profiles/*profile\_name* directory.

#### *app\_server\_root*

The default installation root directory for WebSphere Application Server is the /QIBM/ProdData/WebSphere/AppServer/V7/Base directory.

#### *cip\_app\_server\_root*

The default installation root directory is the /QIBM/ProdData/WebSphere/AppServer/V7/Base/cip/*cip\_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

A CIP is a WebSphere Application Server product bundled with optional maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts.

#### *cip\_profile\_root*

The default profile root directory is the /QIBM/UserData/WebSphere/AppServer/V7/Base/cip/*cip\_uid*/profiles/*profile\_name* directory for a customized installation package (CIP) produced by the Installation Factory.

#### *cip\_user\_data\_root*

The default user data root directory is the /QIBM/UserData/WebSphere/AppServer/V7/Base/cip/*cip\_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

*if\_root* This directory represents the root directory of the IBM WebSphere Installation Factory. Because you can download and unpack the Installation Factory to any directory on the file system to which you have write access, this directory's location varies by user. The Installation Factory is an Eclipse-based tool which creates installation packages for installing WebSphere Application Server in a reliable and repeatable way, tailored to your specific needs.

#### *iip\_root*

This directory represents the root directory of an *integrated installation package* (IIP) produced by the IBM WebSphere Installation Factory. Because you can create and save an IIP to any directory on the file system to which you have write access, this directory's location varies by user. An IIP is an aggregated installation package created with the Installation Factory that can include one or more generally available installation packages, one or more customized installation packages (CIPs), and other user-specified files and directories.

#### *java\_home*

The following directories are the root directories for all supported Java Virtual Machines (JVMs).

| JVM                            | Directory                                  |
|--------------------------------|--------------------------------------------|
| Classic JVM                    | /QIBM/ProdData/Java400/jdk6                |
| 32-bit IBM Technology for Java | /QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit |
| 64-bit IBM Technology for Java | /QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit |

#### *plugins\_profile\_root*

The default Web server plug-ins profile root is the /QIBM/UserData/WebSphere/Plugins/V7/webserver/profiles/*profile\_name* directory.

#### *plugins\_root*

The default installation root directory for Web server plug-ins is the /QIBM/ProdData/WebSphere/Plugins/V7/webserver directory.

#### *plugins\_user\_data\_root*

The default Web server plug-ins user data root is the /QIBM/UserData/WebSphere/Plugins/V7/webserver directory.

#### *product\_library*

##### *product\_lib*

This is the product library for the installed product. The product library for each Version 7.0 installation on the system contains the program and service program objects (similar to .exe, .dll, .so objects) for the installed product. The product library name is QWAS7x (where x is A, B, C, and so on). The product library for the first WebSphere Application Server Version 7.0 product installed on the system is QWAS7A. The *app\_server\_root*/properties/product.properties file contains the value for the product library of the installation, was.install.library, and is located under the *app\_server\_root* directory.

#### *profile\_root*

The default directory for a profile named *profile\_name* for WebSphere Application Server is the /QIBM/UserData/WebSphere/AppServer/V7/Base/profiles/*profile\_name* directory.

#### *shared\_product\_library*

The shared product library, which contains all of the objects shared by all installations on the system, is QWAS7. This library contains objects such as the product definition, the subsystem description, the job description, and the job queue.

#### *updi\_root*

The default installation root directory for the Update Installer for WebSphere Software is the /QIBM/ProdData/WebSphere/UpdateInstaller/V7/updi directory.

#### *user\_data\_root*

The default user data directory for WebSphere Application Server is the /QIBM/UserData/WebSphere/AppServer/V7/Base directory.

The profiles and profileRegistry subdirectories are created under this directory when you install the product.

#### *web\_server\_root*

The default web server path is /www/*web\_server\_name*.

---

## Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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

IBM Director of Intellectual Property & Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
USA



---

## Trademarks and service marks

IBM, the IBM logo, and [ibm.com](http://ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site ([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.