IBM WebSphere Application Server for Distributed
Platforms, Version 8.5

Administering applications and their environment

IBM

Compilation date: June 12, 2012

Contents

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 email form appears.
    3. Fill out the email form as instructed, and submit your feedback.
- To send comments on PDF books, you can email your comments to: **wasdoc@us.ibm.com**.

    Your comment should pertain to specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. 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.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer. When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about your comments.

Using this PDF

Links

Because the content within this PDF is designed for an online information center deliverable, you might experience broken links. You can expect the following link behavior within this PDF:

- 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.

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.

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

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

> See also Introduction: Environment and Introduction: Variables.

Chapter 2. Administering the Liberty profile

You can administer the Liberty profile from the command prompt, configure it with web server plugins, and capture its status. You can package a Liberty server configuration along with the applications that it runs, for distribution to colleagues, or installation on other systems. If available, you can use the Equinox OSGi console to aid with debugging.

Procedure

- Edit the Liberty profile using developer tools.
- Configure the Liberty profile runtime environment
- Capture the status of a Liberty profile server.
- Package a Liberty profile server.
- Use an OSGi console.

Liberty profile: Configuration elements in the server.xml file

The application server configuration is described in a series of elements in the `server.xml` configuration file. Each element has one or more attributes or sub-elements. This topic contains details of the possible elements, attributes and sub-elements that can be configured.

List of elements in the server.xml configuration file.

- "webContainer" on page 119
- "wlmClassification" on page 125
- "zosWorkloadManager" on page 125

activedLdapFilterProperties

Specifies the list of default Active Directory LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.actived.

userFilter
    An LDAP filter clause for searching the user registry for users.

    string

    (&(sAMAccountName=%v)(objectcategory=user))

    true

groupFilter
    An LDAP filter clause for search the user registry for groups.

    string

    (&(cn=%v)(objectcategory=group))

    true

userIdMap
    An LDAP filter that maps the name of a user to an LDAP entry.

    string

    user:sAMAccountName

    true

groupIdMap
    groupIdFilter.desc

    string

    *:cn

    true

groupMemberIdMap
    An LDAP filter that identifies user to group memberships.

    string

    memberof:member

    true

administrator-role

A collection of users and/or groups assigned the server administrator role. PID is com.ibm.ws.management.security.role.administrator.

user
    User assigned a role.

    string

    false

group
    Group assigned a role.

string

false

application

Defines the properties of an application. PID is com.ibm.ws.app.manager.

location
    Location of an application expressed as an absolute path or a path relative to the server-level apps directory.

    string

    true

name
    Name of an application.

    string

    false

type
    Type of application archive.

    string

    false

context-root
    Context root of an application.

    string

    false

autoStart
    Indicates whether or not the server should start the application automatically when the server starts.

    boolean

    true

    false

application-bnd

Binds general deployment information included in the application to specific resources. PID is com.ibm.ws.javaee.dd.appbnd, and it is the child of complex type "application".

version
    Version of the application bindings specification.

    string

    false

security-role
    A role that is mapped to users and groups in a domain user registry.

    name
        Name of a security role.

        string

        true

user
> A user possessing a security role.
>
> **name**
> > Name of a user possessing a security role.
> >
> > string
> >
> > true
>
> **access-id**
> > A user access ID in the general form user:realmName/userUniqueId. A value will be generated if one is not specified.
> >
> > string
> >
> > false
>
> false

group
> A group possessing a security role.
>
> **name**
> > Name of a group possessing a security role.
> >
> > string
> >
> > true
>
> **access-id**
> > Group access ID
> >
> > string
> >
> > false
>
> false

special-subject
> Name of a special-subject possessing a security role.
>
> **type**
> > One of the following special subject types: ALL_AUTHENTICATED_USERS, EVERYONE.
> >
> > string
> >
> > **EVERYONE**
> >
> > **ALL_AUTHENTICATED_USERS**
> > > All authenticated users
> >
> > true
>
> false

run-as
> ID and password of a user required to access a bean from another bean.
>
> **userid**
> > ID of a user required to access a bean from another bean.
> >
> > string
> >
> > true

password

Password of a user required to access a bean from another bean. The value can be stored in clear text or encoded form. To encode the password, use the securityUtility tool with the encode option.

password (string)

false

false

false

applicationMonitor

Defines how the server responds to application additions, updates, and deletions. PID is com.ibm.ws.app.manager.monitor.

pollingRate

Rate at which the server checks for application additions, updates, and deletions. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

500ms

false

dropins

Location of the application drop-in directory expressed as an absolute path or a path relative to the server directory.

string

dropins

false

dropinsEnabled

Monitor the drop-in directory for application additions, updates, and deletions.

boolean

true

false

updateTrigger

Application update method or trigger.

string

polled

polled Server will scan for application changes at the polling interval and update any applications that have detectable changes.

mbean

Server will only update applications when prompted by an MBean called by an external program such as an integrated development environment or a management application.

disabled

Disables all update monitoring. Application changes will not be applied while the server is running.

false

authCache

Controls the operation of the authentication cache service. PID is
com.ibm.ws.security.authentication.cache.

`initialSize`
Initial number of entries supported by the authentication cache.

int

50

false

`maxSize`
Maximum number of entries supported by the authentication cache.

int

25000

false

`timeout`
Amount of time after which an entry in the cache will be removed. Specify a positive integer followed
by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example,
specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example,
1s500ms is equivalent to 1.5 seconds.

string

600s

false

`allowBasicAuthLookup`
Allow lookup by user ID and hashed password.

boolean

true

false

authData

Authentication data for a JCA connection to an Enterprise Information System (EIS). PID is
com.ibm.ws.security.jca.internal.authdata.config.

`user`
Name of the user to use when connecting to the EIS.

string

true

`password`
Password of the user to use when connecting to the EIS. The value can be stored in clear text or
encoded form. It is recommended that you encode the password. To do so, use the securityUtility tool
with the encode option.

password (string)

true

authentication

Controls the built-in authentication service configuration. PID is com.ibm.ws.security.authentication.

`cacheEnabled`

> Enables the authentication cache.
>
> boolean
>
> true
>
> true

`allowHashtableLoginWithIdOnly`

> Allow an application to login with just an identity in the hashtable properties. Use this option only when you have applications that require this and have other means to validate the identity.
>
> boolean
>
> false
>
> false

basicRegistry

A simple XML-based user registry. PID is com.ibm.ws.security.registry.basic.config.

`realm`

> The realm name represents the user registry.
>
> string
>
> BasicRegistry
>
> true

`user`

> A user in a Basic User Registry.
>
> **`name`**
>
> > Name of a user in a Basic User Registry.
> >
> > string
> >
> > true
>
> **`password`**
>
> > Password of a user in a Basic User Registry. The value can be stored in clear text or encoded form. It is recommended that you encode the password. To do so, use the securityUtility tool with the encode option.
> >
> > password (string)
> >
> > true
>
> false

`group`

> A group in a Basic User Registry.
>
> **`name`**
>
> > Name of a group in a Basic User Registry.
> >
> > string
> >
> > true
>
> **`member`**
>
> > A member of a Basic User Registry group.

name
>> Name of a user in a Basic User Registry group.

>> string

>> true

> false

> false

bundleRepository

EBA bundle repository service. PID is com.ibm.ws.eba.bundle.repository.

filesetRef
> Space separated list of fileset references

> List of configuration IDs of type fileset (comma-separated string).

> false

fileset
> Space separated list of fileset references

> Element of type fileset.

> false

channelfw

Defines channel and chain management settings. PID is com.ibm.ws.channelfw.

chainStartRetryInterval
> Time interval between start retries. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

> string

> 5s

> false

chainStartRetryAttempts
> Number of retry attempts to make per chain.

> int

> 60

> false

chainQuiesceTimeout
> Default amount of time to wait while quiescing chains. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

> string

> 30s

> false

warningWaitTime
> Amount of time to wait before notifying of a missing factory configuration. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
>
> string
>
> 10s
>
> false

classloader

Classloader Service. PID is com.ibm.ws.classloading.classloader, and it is the child of complex type "application".

delegation
> Controls whether parent classloader is used before or after this classloader.
>
> string
>
> parentFirst
>
> **parentFirst**
>>   Parent first
>
> **parentLast**
>>   Parent last
>
> false

privateLibraryRef
> List of library references. Library class instances are unique to this classloader, independent of class instances from other classloaders.
>
> List of configuration IDs of type library (comma-separated string).
>
> false

commonLibraryRef
> List of library references. Library class instances are shared with other classloaders.
>
> List of configuration IDs of type library (comma-separated string).
>
> false

apiTypeVisibility
> The types of API package this class loader will be able to see, as a comma-separated list of any combination of the following: spec, ibm-api, ibm-spi, third-party.
>
> string
>
> spec,ibm-api
>
> false

privateLibrary
> List of library references. Library class instances are unique to this classloader, independent of class instances from other classloaders.
>
> Element of type library.
>
> false

commonLibrary
> List of library references. Library class instances are shared with other classloaders.

Element of type library.

false

config

Defines how the server processes configuration information. PID is com.ibm.ws.config.

onError
Action to take after a incurring a configuration error.

onError

string

WARN

WARN
Server will issue warning and error messages when it incurs a configuration error.

FAIL Server will issue a warning or error message on the first error occurrence and then stop the server.

IGNORE
Server will not issue any warning and error messages when it incurs a configuration error.

true

monitorInterval
Rate at which the server checks for configuration updates. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

500ms

false

updateTrigger
Configuration update method or trigger.

string

polled

polled Server will scan for changes at the polling interval on all the configuration files and update the runtime configuration with the changes detected.

mbean
Server will only update the configuration when prompted by an MBean called by an external program such as an integrated development environment or a management application.

disabled
Disables all update monitoring. Configuration changes will not be applied while the server is running.

false

connectionManager

Connection Manager configuration. PID is com.ibm.ws.jca.connectionManager.

agedTimeout
Amount of time before a physical connection can be discarded by pool maintenance. A value of -1 disables this timeout. Specify a positive integer followed by a unit of time, which can be hours (h),

minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

-1

false

`connectionTimeout`
Amount of time after which a connection request times out. A value of -1 disables this timeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

30s

false

`maxIdleTime`
Amount of time after which an unused or idle connection can be discarded during pool maintenance, if doing so does not reduce the pool below the minimum size. A value of -1 disables this timeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

30m

false

`maxPoolSize`
Maximum number of physical connections for a pool. A value of 0 means unlimited.

int

50

false

`minPoolSize`
Minimum number of physical connections to maintain in the pool. The pool is not pre-populated. Aged timeout can override the minimum.

int

false

`purgePolicy`
Specifies which connections to destroy when a stale connection is detected in a pool.

string

EntirePool

EntirePool
When a stale connection is detected, all connections in the pool are marked stale, and when no longer in use, are closed.

FailingConnectionOnly
When a stale connection is detected, only the connection which was found to be bad is closed.

ValidateAllConnections
>
>> When a stale connection is detected, connections are tested and those found to be bad are closed.
>
> false

reapTime
>
> Amount of time between runs of the pool maintenance thread. A value of -1 disables pool maintenance. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.
>
> string
>
> 3m
>
> false

maxConnectionsPerThread
>
> Limits the number of open connections on each thread.
>
> int
>
> false

numConnectionsPerThreadLocal
>
> Caches the specified number of connections for each thread.
>
> int
>
> false

customLdapFilterProperties

Specifies the list of default Custom LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.custom.

userFilter
>
> An LDAP filter clause for searching the user registry for users.
>
> string
>
> (&(uid=%v)(objectclass=ePerson))
>
> true

groupFilter
>
> An LDAP filter clause for search the user registry for groups.
>
> string
>
> (&(cn= %v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))
>
> true

userIdMap
>
> An LDAP filter that maps the name of a user to an LDAP entry.
>
> string
>
> *:uid
>
> true

groupIdMap
>
> groupIdFilter.desc
>
> string

*:cn

true

groupMemberIdMap
An LDAP filter that identifies user to group memberships.

string

ibm-allGroups:member;ibm-allGroups:uniqueMember;groupOfNames:member;groupOfUniqueNames:uniqueMember

true

dataSource

Defines a data source configuration. PID is com.ibm.ws.jdbc.dataSource.

jndiName
JNDI name for a data source.

string

true

jdbcDriverRef
JDBC driver for a data source.

Configuration ID of type jdbcDriver (string).

false

connectionManagerRef
Connection manager for a data source.

Configuration ID of type connectionManager (string).

false

type
Type of data source.

string

javax.sql.XADataSource

javax.sql.ConnectionPoolDataSource

javax.sql.DataSource

false

connectionSharing
Specifies how connections are matched for sharing.

string

MatchOriginalRequest

MatchOriginalRequest
When sharing connections, match based on the original connection request.

MatchCurrentState
When sharing connections, match based on the current state of the connection.

false

isolationLevel
Default transaction isolation level.

string

TRANSACTION_READ_UNCOMMITTED
Dirty reads, non-repeatable reads and phantom reads can occur.

TRANSACTION_READ_COMMITTED
Dirty reads are prevented; non-repeatable reads and phantom reads can occur.

TRANSACTION_REPEATABLE_READ
Dirty reads and non-repeatable reads are prevented; phantom reads can occur.

TRANSACTION_SERIALIZABLE
Dirty reads, non-repeatable reads and phantom reads are prevented.

TRANSACTION_SNAPSHOT
Snapshot isolation for Microsoft SQL Server JDBC Driver and DataDirect Connect for JDBC driver.

false

statementCacheSize
Maximum number of cached statements per connection.

int

10

false

transactional
Enable participation in transactions that are managed by the application server.

boolean

true

false

beginTranForResultSetScrollingAPIs
Attempt transaction enlistment when result set scrolling interfaces are used.

boolean

true

false

beginTranForVendorAPIs
Attempt transaction enlistment when vendor interfaces are used.

boolean

true

false

commitOrRollbackOnCleanup
Determines how to clean up connections that might be in a database unit of work (AutoCommit=false) when the connection is closed or returned to the pool.

string

commit
Clean up the connection by committing.

rollback
Clean up the connection by rolling back.

false

queryTimeout

> Default query timeout for SQL statements. In a JTA transaction, syncQueryTimeoutWithTransactionTimeout can override this default. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.
>
> string
>
> false

recoveryAuthDataRef

> Authentication data for transaction recovery.
>
> Configuration ID of type authData (string).
>
> false

syncQueryTimeoutWithTransactionTimeout

> Use the time remaining (if any) in a JTA transaction as the default query timeout for SQL statements.
>
> boolean
>
> false
>
> false

supplementalJDBCTrace

> Supplements the JDBC driver trace that is logged when JDBC driver trace is enabled in bootstrap.properties. JDBC driver trace specifications include: com.ibm.ws.database.logwriter, com.ibm.ws.db2.logwriter, com.ibm.ws.derby.logwriter, com.ibm.ws.informix.logwriter, com.ibm.ws.oracle.logwriter, com.ibm.ws.sqlserver.logwriter, com.ibm.ws.sybase.logwriter.
>
> boolean
>
> false

jdbcDriver

> JDBC driver for a data source.
>
> Element of type jdbcDriver.
>
> false

connectionManager

> Connection manager for a data source.
>
> Element of type connectionManager.
>
> false

recoveryAuthData

> Authentication data for transaction recovery.
>
> Element of type authData.
>
> false

domino50LdapFilterProperties

Specifies the list of default Domino® LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.domino50.

userFilter

> An LDAP filter clause for searching the user registry for users.
>
> string

(&(uid=%v)(objectclass=Person))

        true

`groupFilter`
        An LDAP filter clause for search the user registry for groups.

        string

        (&(cn=%v)(objectclass=dominoGroup))

        true

`userIdMap`
        An LDAP filter that maps the name of a user to an LDAP entry.

        string

        person:uid

        true

`groupIdMap`
        groupIdFilter.desc

        string

        *:cn

        true

`groupMemberIdMap`
        An LDAP filter that identifies user to group memberships.

        string

        dominoGroup:member

        true

edirectoryLdapFilterProperties
Specifies the list of EDirectory LDAP filters. PID is
com.ibm.ws.security.registry.ldap.internal.filters.eDirectory.

`userFilter`
        An LDAP filter clause for searching the user registry for users.

        string

        (&(cn=%v)(objectclass=Person))

        true

`groupFilter`
        An LDAP filter clause for search the user registry for groups.

        string

        (&(cn=%v)(objectclass=groupOfNames))

        true

`userIdMap`
        An LDAP filter that maps the name of a user to an LDAP entry.

        string

        person:cn

        true

groupIdMap

    groupIdFilter.desc

    string

    *:cn

    true

groupMemberIdMap

    An LDAP filter that identifies user to group memberships.

    string

    groupOfNames:member

    true

executor

Defines threading and execution settings for the server. PID is com.ibm.ws.threading.

name

    Name of the executor for which the thread is performing work.

    string

    Default Executor

    false

maxThreads

    Maximum number of threads that can be associated with the executor. If greater than 0, this value must be greater than or equal to the value of coreThreads. If the value of maxThreads is less than or equal to 0, the maximum number of threads is unbounded.

    int

    -1

    false

coreThreads

    Steady-state or core number of threads to associate with the executor. The number of threads associated with the executor will quickly grow to this number. If this value is less than 0, a default value is used. This default value is calculated based on the number of hardware threads on the system.

    int

    -1

    false

keepAlive

    Amount of time to keep an idle thread in the pool before allowing it to terminate. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

    string

    60s

    false

stealPolicy
> The work-stealing policy to employ. The options for this policy determine how work is queued, and how threads obtain queued work.
>
> string
>
> STRICT
>
> **STRICT**
> > All threads that generate work own a local work pile. Threads that are associated with the executor take work from other threads when the local work pile is exhausted.
>
> **LOCAL**
> > A global work queue is used for work that is generated by threads that are not associated with the executor. Work generated by threads associated with the executor are placed on a local work pile. This work pile is owned by the generating thread, unless another thread steals it. Threads that are associated with the executor take work associated with other threads if the local work pile is empty and there is no work on the global work queue.
>
> **NEVER**
> > A global work queue is used to feed work to threads that are associated with the executor. No stealing will occur.
>
> false

rejectedWorkPolicy
> Policy to employ when the executor is unable to stage work for execution.
>
> string
>
> ABORT
>
> **ABORT**
> > Raise an exception.
>
> **CALLER_RUNS**
> > Execute the work immediately on the caller's thread.
>
> false

featureManager

Defines how the server loads features. PID is com.ibm.ws.kernel.feature.

onError
> Action to take after a failure to load a feature.
>
> onError
>
> string
>
> WARN
>
> **WARN**
> > Server will issue warning and error messages when it incurs a feature configuration error.
>
> **FAIL** Server will issue a warning or error message on the first feature configuration error occurrence and then stop the server.
>
> **IGNORE**
> > Server will not issue any warning and error messages when it incurs a feature configuration error.
>
> true

feature
> string

false

fileset

Fileset Service. PID is com.ibm.ws.kernel.metatype.helper.fileset.

dir
    The base directory to search for files.

    string

    .

    true

caseSensitive
    Boolean to indicate whether or not the search should be case sensitive (default: true).

    boolean

    true

    false

includes
    The comma or space separated list of file name patterns to include in the search results (default: *).

    string

    *

    false

excludes
    The comma or space separated list of file name patterns to exclude from the search results, by default no files are excluded.

    string

    false

scanInterval
    Scanning interval to check the fileset for changes as a long with a time unit suffix h-hour, m-minute, s-second, ms-millisecond (e.g. 2ms or 5s). Disabled (scanInterval=0) by default. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

    string

    0

    false

httpClassification

zos.wlm.httpclassification.config.description. PID is com.ibm.ws.zos.wlm.httpclassification, and it is the child of complex type "wlmClassification".

transactionClass
    Defines the priority

    string

    false

host
    Defines which host to map the transaction class to

string

*

false

port
Defines which port to map the transaction class to

string

*

false

resource
Defines the URI to use when mapping transaction class

string

*

false

method
Defines the HTTP method to map to

string

*

false

httpEncoding

Configuration properties for the HTTP Transport Encoding service. PID is com.ibm.ws.transport.http.encoding.

converter.Shift_JIS
Converter for Shift_JIS

string

Cp943C

false

converter.EUC-JP
Converter for EUC-JP

string

Cp33722C

false

converter.EUC-KR
Converter for EUC-KR

string

Cp970

false

converter.EUC_KR
Converter for EUC_KR

string

Cp970

false

converter.EUC-TW
Converter for EUC-TW

string

Cp964

false

converter.Big5
Converter for Big5

string

Cp950

false

converter.GB2312
Converter for GB2312

string

EUC_CN

false

converter.ISO-2022-KR
Converter for ISO-2022-KR

string

ISO2022KR

false

encoding.en
Encoding for 'en' locale

string

ISO-8859-1

false

encoding.fr
Encoding for 'fr' locale

string

ISO-8859-1

false

encoding.de
Encoding for 'de' locale

string

ISO-8859-1

false

encoding.es
Encoding for 'es' locale

string

ISO-8859-1

false

encoding.pt
Encoding for 'pt' locale

string

ISO-8859-1

false

encoding.da
Encoding for 'da' locale

string

ISO-8859-1

false

encoding.ca
Encoding for 'ca' locale

string

ISO-8859-1

false

encoding.fi
Encoding for 'fi' locale

string

ISO-8859-1

false

encoding.it
Encoding for 'it' locale

string

ISO-8859-1

false

encoding.nl
Encoding for 'nl' locale

string

ISO-8859-1

false

encoding.no
Encoding for 'no' locale

string

ISO-8859-1

false

encoding.sv
Encoding for 'sv' locale

string

ISO-8859-1

false

encoding.is
Encoding for 'is' locale

string

ISO-8859-1

false

encoding.eu
Encoding for 'eu' locale

string

ISO-8859-1

false

encoding.cs
Encoding for 'cs' locale

string

ISO-8859-2

false

encoding.hr
Encoding for 'hr' locale

string

ISO-8859-2

false

encoding.hu
Encoding for 'hu' locale

string

ISO-8859-2

false

encoding.lt
Encoding for 'lt' locale

string

ISO-8859-2

false

encoding.pl
Encoding for 'pl' locale

string

ISO-8859-2

false

encoding.sh
Encoding for 'sh' locale

string

ISO-8859-2

false

encoding.sk
Encoding for 'sk' locale

string

ISO-8859-2

false

encoding.sl
Encoding for 'sl' locale

string

ISO-8859-2

false

encoding.sq
Encoding for 'sq' locale

string

ISO-8859-2

false

encoding.fo
Encoding for 'fo' locale

string

ISO-8859-2

false

encoding.ro
Encoding for 'ro' locale

string

ISO-8859-2

false

encoding.mt
Encoding for 'mt' locale

string

ISO-8859-3

false

encoding.et
Encoding for 'et' locale

string

ISO-8859-4

false

encoding.lv
Encoding for 'lv' locale

string

ISO-8859-4

false

encoding.be
Encoding for 'be' locale

string

ISO-8859-5

false

encoding.bg
Encoding for 'bg' locale

string

ISO-8859-5

false

encoding.mk
Encoding for 'mk' locale

string

ISO-8859-5

false

encoding.ru
Encoding for 'ru' locale

string

ISO-8859-5

false

encoding.sr
Encoding for 'sr' locale

string

ISO-8859-5

false

encoding.uk
Encoding for 'uk' locale

string

ISO-8859-5

false

encoding.ar
Encoding for 'ar' locale

string

ISO-8859-6

false

encoding.fa
Encoding for 'fa' locale

string

ISO-8859-6

false

encoding.ms
Encoding for 'ms' locale

string

ISO-8859-6

false

encoding.el
Encoding for 'el' locale

string

ISO-8859-7

false

encoding.iw
Encoding for 'iw' locale

string

ISO-8859-8

false

encoding.he
Encoding for 'he' locale

string

ISO-8859-8

false

encoding.ji
Encoding for 'ji' locale

string

ISO-8859-8

false

encoding.yi
Encoding for 'yi' locale

string

ISO-8859-8

false

encoding.tr
Encoding for 'tr' locale

string

ISO-8859-9

false

encoding.th
Encoding for 'th' locale

string

windows-874

false

encoding.vi
Encoding for 'vi' locale

string

windows-1258

false

encoding.ja
Encoding for 'ja' locale

string

Shift_JIS

false

encoding.ko
Encoding for 'ko' locale

string

EUC-KR

false

encoding.zh
Encoding for 'zh' locale

string

GB2312

false

encoding.zh_TW
Encoding for 'zh_TW' locale

string

Big5

false

encoding.hy
Encoding for 'hy' locale

string

UTF-8

false

encoding.ka
Encoding for 'ka' locale

string

UTF-8

false

encoding.hi
Encoding for 'hi' locale

string

UTF-8

false

encoding.mr
    Encoding for 'mr' locale

    string

    UTF-8

    false

encoding.sa
    Encoding for 'sa' locale

    string

    UTF-8

    false

encoding.ta
    Encoding for 'ta' locale

    string

    UTF-8

    false

encoding.bn
    Encoding for 'bn' locale

    string

    UTF-8

    false

httpEndpoint

Configuration properties for an HTTP endpoint. PID is com.ibm.ws.http.

httpOptionsRef
    HTTP protocol options for the endpoint.

    defaultHTTPVar

    Configuration ID of type httpOptions (string).

    false

sslOptionsRef
    SSL protocol options for the endpoint.

    defaultSSLVar

    Configuration ID of type sslOptions (string).

    false

tcpOptionsRef
    TCP protocol options for the endpoint.

    Configuration ID of type tcpOptions (string).

    defaultTCPOptions

    false

enabled

Toggle the availability of an endpoint. When true, this endpoint will be activated by the dispatcher to handle HTTP requests.

boolean

true

false

host

IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name, used by a client to request a resource. Use '*' for all available network interfaces.

string

localhost

false

httpPort

The port used for client HTTP requests. Use -1 to disable this port.

int

false

httpsPort

The port used for client HTTP requests secured with SSL (https). Use -1 to disable this port.

int

false

virtualHost

ID of a configured virtual host. All endpoints are associated with the 'default_host' virtual host by default.

string

default_host

false

httpOptions

HTTP protocol options for the endpoint.

defaultHTTPVar

Element of type httpOptions.

false

sslOptions

SSL protocol options for the endpoint.

defaultSSLVar

Element of type sslOptions.

false

tcpOptions

TCP protocol options for the endpoint.

Element of type tcpOptions.

defaultTCPOptions

false

httpOptions

HTTP protocol configuration. PID is com.ibm.ws.http.options.

keepAliveEnabled
    Enables persistent connections (HTTP keepalive). If true, connections are kept alive for reuse by multiple sequential requests and responses. If false, connections are closed after the response is sent.

    boolean

    true

    false

maxKeepAliveRequests
    Maximum number of persistent requests that are allowed on a single HTTP connection if persistent connections are enabled. A value of -1 means unlimited.

    int

    100

    false

persistTimeout
    Amount of time that a socket will be allowed to remain idle between requests. This setting only applies if persistent connections are enabled. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

    string

    30s

    false

readTimeout
    Amount of time to wait for a read request to complete on a socket after the first read occurs. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

    string

    60s

    false

writeTimeout
    Amount of time to wait on a socket for each portion of the response data to be transmitted. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

    string

    60s

    false

httpSession

Configuration for HTTP session management. PID is com.ibm.ws.session.

storageRef
    The ID of the persistent storage location for session data. If no storage location is defined, session data will be stored in the local application server's memory.

string

false

sslTrackingEnabled
Specifies that session tracking uses Secure Sockets Layer (SSL) information as a session ID.

boolean

false

false

urlRewritingEnabled
Specifies that the session management facility uses rewritten URLs to carry the session IDs.

boolean

false

false

protocolSwitchRewritingEnabled
Adds the session ID to a URL when the URL requires a switch from HTTP to HTTPS or from HTTPS to HTTP.

boolean

false

false

cookiesEnabled
Specifies that session tracking uses cookies to carry session IDs.

boolean

true

false

cookieName
A unique name for a session management cookie.

string

JSESSIONID

false

cookieDomain
Domain field of a session tracking cookie.

string

false

cookieMaxAge
Maximum amount of time that a cookie can reside on the client browser.

int

-1

false

cookiePath
A cookie is sent to the URL designated in the path.

string

/

false

cookieSecure
Specifies that the session cookies include the secure field.

boolean

false

false

cookieHttpOnly
Specifies that session cookies include the HttpOnly field. Browsers that support the HttpOnly field do not enable cookies to be accessed by client-side scripts. Using the HttpOnly field will help prevent cross-site scripting attacks.

boolean

true

false

maxInMemorySessionCount
Maximum number of sessions to maintain in memory for each web module.

int

1000

false

allowOverflow
Allows the number of sessions in memory to exceed the value of the Max in-memory session count property.

boolean

true

false

invalidationTimeout
Amount of time a session can go unused before it is no longer valid.

long

1800

false

securityIntegrationEnabled
Enables security integration, which causes the session management facility to associate the identity of users with their HTTP sessions.

boolean

true

false

idLength
Length of the session identifier.

int

23

false

useContextRootAsCookiePath
>   Specifies that the cookie path equals the context root of the web module instead of /

>   boolean

>   false

>   false

alwaysEncodeUrl
>   The Servlet 2.5 specification specifies to not encode the URL on a response.encodeURL call if it is not necessary. To support backward compatibility when URL encoding is enabled, set this property to true to call the encodeURL method. The URL is always encoded, even if the browser supports cookies.

>   boolean

>   false

>   false

cloneId
>   The clone ID of the cluster member. Within a cluster, this ID must be unique to maintain session affinity. When set, this name overwrites the default name generated by the server.

>   string

>   false

cloneSeparator
>   The single character used to separate the session ID from the clone ID in session cookies. The default value should usually be used. On some Wireless Application Protocol (WAP) devices, a colon (:) is not allowed, so a plus sign (+) should be used instead. Different values should rarely be used. You should understand the clone character requirements of other products running on your system before using this property to change the clone separator character. The fact that any character can be specified as the value for this property does not imply that the character you specify will function correctly. This fact also does not imply that IBM® is responsible for fixing any problem that might arise from using an alternative character.

>   string

>   :

>   false

debugCrossover
>   Enable this option to perform additional checks to verify that only the session associated with the request is accessed or referenced, and log messages if any discrepancies are detected. Disable this option to skip the additional checks.

>   boolean

>   false

>   false

forceInvalidationMultiple
>   If your requests normally are not bound by a response time limit, specify 0 to indicate that the session manager should wait indefinitely until a request is complete before attempting to invalidate the session. Otherwise, set this property to a positive integer to delay the invalidation of active sessions. Active timed out sessions will not be invalidated by the first invalidation interval pass, but will be invalidated by the interval pass based on this value. For example, a value of 2 would invalidate an active session on the second invalidation interval pass after the session timeout has expired.

>   int

>   3

false

idReuse

In a multi-JVM environment that is not configured for session persistence, setting this property to "true" enables the session manager to use the same session information for all of a user's requests even if the web applications that are handling these requests are governed by different JVMs. The default value for this property is false. Set this property to true if you want to enable the session manager to use the session ID sent from a browser to preserve session data across web applications that are running in an environment that is not configured for session persistence.

boolean

false

false

noAdditionalInfo

Forces removal of information that is not needed in session identifiers.

boolean

false

false

reaperPollInterval

The wake-up interval, in seconds, for the process that removes invalid sessions. The minimum value is 30 seconds. If a value less than the minimum is entered, an appropriate value is automatically determined and used. This value overrides the default installation value, which is between 30 and 360 seconds, based off the session timeout value. Because the default session timeout is 30 minutes, the reaper interval is usually between 2 and 3 minutes.

long

-1

false

rewriteId

Use this property to change the key used with URL rewriting.

string

jsessionid

false

securityUserIgnoreCase

Indicates that the session security identity and the client security identity should be considered a match even if their cases are different. For example, when this property is set to true, the session security identity USER1 matches the client security identities User1 and user1.

boolean

false

false

httpSessionDatabase

Controls how HTTP sessions are persisted to a database. PID is com.ibm.ws.session.db.

dataSourceRef

The ID of the data source the session manager should use to persist HTTP session data.

string

true

useMultiRowSchema

When enabled, each session data attribute is placed in a separate row in the database, allowing larger amounts of data to be stored for each session. This configuration can yield better performance when session attributes are very large and few changes are required to the session attributes. When disabled, all session data attributes are placed in the same row for each session.

boolean

false

false

db2RowSize

Table space page size configured for the sessions table, if using a DB2® database. Increasing this value can improve database performance in some environments.

string

4KB

4KB   Use the default table space page size of 4 KB. You do not need to create a DB2 buffer pool or table space, and you do not need to specify a table space name.

8KB   Use a table space page size of 8 KB. You must additionally create a DB2 buffer pool and table space, and specify 8KB as the page size for both. You must also specify the name of the table space you created.

16KB   Use a table space page size of 16 KB. You must additionally create a DB2 buffer pool and table space, and specify 16KB as the page size for both. You must also specify the name of the table space you created.

32KB   Use a table space page size of 32 KB. You must additionally create a DB2 buffer pool and table space, and specify 32KB as the page size for both. You must also specify the name of the table space you created.

false

tableSpaceName

Table space to be used for the sessions table. This value is only required when the DB2 Row Size is greater than 4KB.

string

false

scheduleInvalidation

Enable this option to reduce the number of database updates required to keep the HTTP sessions alive. Specify the two hours of a day when there is the least activity in the application server. When this option is disabled, the invalidator process runs every few minutes to remove invalidated HTTP sessions.

boolean

false

false

scheduleInvalidationFirstHour

Indicates the first hour during which the invalidated sessions are cleared from the persistent store. Specify this value as an integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

int

0

false

scheduleInvalidationSecondHour
>Indicates the second hour during which the invalidated sessions are cleared from the persistent store. Specify this value as an integer between 0 and 23. This value is valid only when schedule invalidation is enabled.
>
>int
>
>0
>
>false

writeFrequency
>Specifies when session data is written to the persistent store. By default, session data is written to the persistent store after the servlet completes execution. Changing this value can improve performance in some environments.
>
>string
>
>END_OF_SERVLET_SERVICE
>
>**END_OF_SERVLET_SERVICE**
>>Session data is written to the persistent store after the servlet completes execution.
>
>**MANUAL_UPDATE**
>>A programmatic sync on the IBMSession object is required to write the session data to the persistent store.
>
>**TIME_BASED_WRITE**
>>Session data is written to the persistent store based on the specified write interval value.
>
>false

writeInterval
>Number of seconds that should pass before writing session data to the persistent store. The default is 120 seconds. This value is only used when a time based write frequency is enabled.
>
>int
>
>120
>
>false

writeContents
>Specifies how much session data should be written to the persistent store. By default, only updated attributes are written, but all attributes can be written instead (regardless of whether or not they changed).
>
>string
>
>ONLY_UPDATED_ATTRIBUTES
>
>**ONLY_UPDATED_ATTRIBUTES**
>>Only updated attributes are written to the persistent store.
>
>**ALL_SESSION_ATTRIBUTES**
>>All attributes are written to the persistent store.
>
>false

noAffinitySwitchBack
>Set this property to "true" to maintain affinity to the new member even after original one comes back up. When a cluster member fails, its requests routed to a different cluster member, and sessions are activated in that other member. Thus, session affinity is maintained to the new member, and when failed cluster member comes back up, the requests for sessions that were created in the original cluster member are routed back to it. Allowed values are true or false, with the default being false. Set

this property to true when you have distributed sessions configured with time-based write. Note that this property has no affect on the behavior when distributed sessions is not enabled.

boolean

false

false

onlyCheckInCacheDuringPreInvoke

A value of true indicates that the last access time of a session should only be updated if a request gets the session. A value of false indicates that the last access time of a session should be updated upon every request. Changing this value can improve performance in some environments.

boolean

false

false

optimizeCacheIdIncrements

If the user's browser session is moving back and forth across multiple web applications, you might see extra persistent store activity as the in-memory sessions for a web module are refreshed from the persistent store. As a result, the cache IDs are continually increasing and the in-memory session attributes are overwritten by those of the persistent copy. Set this property to true if you want to prevent the cache IDs from continually increasing. A value of true indicates that the session manager should assess whether the in-memory session for a web module is older than the copy in persistent store. If the configuration is a cluster, ensure that the system times of each cluster member are as identical as possible.

boolean

true

false

tableName

The database table name.

string

sessions

false

useInvalidatedId

Set this property to "true" to reuse the incoming ID if the session with that ID was recently invalidated. This is a performance optimization because it prevents checking the persistent store.

boolean

true

false

useOracleBlob

Set this property to "true" to create the database table using the Binary Large Object (BLOB) data type for the medium column. This value increases performance of persistent sessions when Oracle databases are used. Due to an Oracle restriction, BLOB support requires use of the Oracle Call Interface (OCI) database driver for more than 4000 bytes of data. You must also ensure that a new sessions table is created before the server is restarted by dropping your old sessions table or by changing the datasource definition to reference a database that does not contain a sessions table.

boolean

false

false

idsLdapFilterProperties

Specifies the list of default IBM Directory Server LDAP filters. PID is
com.ibm.ws.security.registry.ldap.internal.filters.ids.

userFilter
An LDAP filter clause for searching the user registry for users.

string

(&(uid=%v)(objectclass=ePerson))

true

groupFilter
An LDAP filter clause for search the user registry for groups.

string

(&(cn=
%v)(l(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))

true

userIdMap
An LDAP filter that maps the name of a user to an LDAP entry.

string

*:uid

true

groupIdMap
groupIdFilter.desc

string

*:cn

true

groupMemberIdMap
An LDAP filter that identifies user to group memberships.

string

ibm-allGroups:member;ibm-
allGroups:uniqueMember;groupOfNames:member;groupOfUniqueNames:uniqueMember

true

iplanetLdapFilterProperties

Specifies the list of default IPlanet LDAP filters. PID is
com.ibm.ws.security.registry.ldap.internal.filters.iPlanet.

userFilter
An LDAP filter clause for searching the user registry for users.

string

(&(uid=%v)(objectclass=inetOrgPerson))

true

groupFilter

An LDAP filter clause for search the user registry for groups.

string

(&(cn=%v)(objectclass=ldapsubentry))

true

userIdMap

An LDAP filter that maps the name of a user to an LDAP entry.

string

inetOrgPerson:uid

true

groupIdMap

groupIdFilter.desc

string

*:cn

true

groupMemberIdMap

An LDAP filter that identifies user to group memberships.

string

nsRole:nsRole

true

jaasLoginContextEntry

The JAAS login context entry configuration. PID is
com.ibm.ws.security.authentication.internal.jaas.jaasLoginContextEntry.

name

Name of a JAAS configuration entry.

string

true

loginModuleRef

A reference to the ID of a JAAS login module.

string

true

jaasLoginModule

A login module in the JAAS configuration. PID is
com.ibm.ws.security.authentication.internal.jaas.jaasLoginModuleConfig.

className

Fully-qualified package name of the JAAS login module class.

string

true

controlFlag

The login module's control flag. Valid values are REQUIRED, REQUISITE, SUFFICIENT, and OPTIONAL.

string

REQUIRED

REQUIRED

This LoginModule is REQUIRED as per the JAAS specification. The LoginModule is required to succeed.

REQUISITE

controlFlag.REQUISITE

SUFFICIENT

This LoginModule is SUFFICIENT as per the JAAS specification. The LoginModule is not required to succeed. If authentication is successful, no other LoginModules will be called and control is returned to the caller.

OPTIONAL

This LoginModule is OPTIONAL as per the JAAS specification. The LoginModule is not required to succeed.

true

libraryRef

A reference to the ID of the shared library configuration.

Configuration ID of type library (string).

false

library

A reference to the ID of the shared library configuration.

Element of type library.

false

options

optionsRef.desc

false

jdbcDriver

Identifies a JDBC driver. PID is com.ibm.ws.jdbc.jdbcDriver.

libraryRef

Identifies JDBC driver JARs and native files.

Configuration ID of type library (string).

false

javax.sql.XADataSource

JDBC driver implementation of javax.sql.XADataSource.

string

false

javax.sql.ConnectionPoolDataSource

JDBC driver implementation of javax.sql.ConnectionPoolDataSource.

string

false

`javax.sql.DataSource`
    JDBC driver implementation of javax.sql.DataSource.

    string

    false

`library`
    Identifies JDBC driver JARs and native files.

    Element of type library.

    false

jndiEntry

jndi.entry.service.desc. PID is com.ibm.ws.jndi.internal.JNDIEntry.

`jndiName`
    The JNDI name to use for this entry.

    string

    true

`value`
    The JNDI value to associate with the name.

    string

    true

jpa

Configuration properties for the Java Persistence API container. PID is com.ibm.ws.jpacomponent.

`defaultJtaDataSourceJndiName`
    Default Java™ Transaction API (JTA) data source JNDI name to be used by applications running in this
    server. By default, data sources are JTA. Only data sources that are transactional are allowed in this
    field.

    string

    false

`defaultNonJtaDataSourceJndiName`
    Default non-transactional data source JNDI name to be used by applications running in this server.
    Only data sources that have been marked as non-transactional are allowed in this field.

    string

    false

`defaultPersistenceProvider`
    Default persistence provider.

    string

    com.ibm.websphere.persistence.PersistenceProviderImpl

    false

`entityManagerPoolCapacity`
    EntityManager pool capacity per PersistenceContext reference. The minimum is 0 and the maximum is
    500.

    int

-1

false

excludedApplication
    An application to be excluded from JPA processing.

    string

    false

jspEngine

JSP 2.2 configuration. PID is com.ibm.ws.jsp.2.2.

disableJspRuntimeCompilation
    Disable compilation of JSPs at runtime.

    boolean

    false

    false

extendedDocumentRoot
    Directory that the JSP engine will search for additional JSP files to serve.

    string

    false

jdkSourceLevel
    Default Java source level for JSPs compiled by the JSP engine.

    string

    15

    13

    14

    15

    16

    false

keepGenerated
    Keep Java source files generated for JSPs.

    boolean

    false

    false

useInMemory
    Generate Java source and classes in memory (without writing to disk).

    boolean

    false

    false

recompileJspOnRestart
    Recompile JSPs after an application is restarted. JSPs are recompiled on first access.

    boolean

false

false

useImplicitTagLibs
Allow JSPs to use jsx and tsx tag libs.

boolean

true

false

disableResourceInjection
Disable injection of resources into JSPs.

boolean

false

false

keyStore

A repository of security certificates used for SSL encryption. PID is com.ibm.ws.ssl.keystore.

password
The password used to load the keystore file. The value can be stored in clear text or encoded form. Use the securityUtility tool to encode the password.

password (string)

false

location
An absolute or relative path to the keystore file. If a relative path is provided, the server will attempt to locate the file in the ${server.config.dir}/resources/security directory. Use the keystore file for a file-based keystore, the keyring name for SAF keyrings, or the device configuration file for hardware cryptography devices. In the SSL minimal configuration, the location of the file is assumed to be ${server.config.dir}/resources/security/key.jks.

string

${server.output.dir}/resources/security/key.jks

false

type
A keystore type supported by the target SDK.

string

jks

false

ldapRegistry

Configuration properties for an LDAP user registry. PID is com.ibm.ws.security.registry.ldap.config.

host
Address of the LDAP server in the form of a IP address or a domain name service (DNS) name.

string

true

port
Port number of the LDAP server.

int

true

baseDN
Base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches in the directory service.

string

true

ldapType
Type of LDAP server to which a connection will be established.

string

Microsoft Active Directory
actived

Custom

IBM Lotus® Domino
domino50

Novell eDirectory
edirectory

IBM Tivoli® Directory Server
ibm_dir_server

Sun Java System Directory Server
iplanet

Netscape Directory Server
netscape

IBM SecureWay Directory Server
secureway

true

realm
The realm name represents the user registry.

string

LdapRegistry

false

bindDN
Distinguished name (DN) for the application server, which is used to bind to the directory service.

string

false

bindPassword
Password for the bind DN. The value can be stored in clear text or encoded form. It is recommended that you encode the password. To do so, use the securityUtility tool with the encode option.

password (string)

false

ignoreCase
Perform a case-insensitive authentication check.

boolean

true

false

recursiveSearch
Performs a nested group search. Select this option only if the LDAP server does not support recursive server-side searches.

boolean

false

false

reuseConnection
Requests the application server to reuse the LDAP server connection.

boolean

true

false

sslEnabled
Indicates whether an SSL connection should be made to the LDAP server.

boolean

false

false

sslRef
ID of the SSL configuration to be used to connect to the SSL-enabled LDAP server.

string

false

searchTimeout
Maximum time for an LDAP server to respond before a request is canceled. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

2m

false

activedFiltersRef
Specifies the list of default Active Directory LDAP filters.

Configuration ID of type activedLdapFilterProperties (string).

false

customFiltersRef
Specifies the list of default Custom LDAP filters.

Configuration ID of type customLdapFilterProperties (string).

false

domino50FiltersRef
Specifies the list of default Domino LDAP filters.

Configuration ID of type domino50LdapFilterProperties (string).

false

edirectoryFiltersRef
Specifies the list of EDirectory LDAP filters.

Configuration ID of type edirectoryLdapFilterProperties (string).

false

idsFiltersRef
Specifies the list of default IBM Directory Server LDAP filters.

Configuration ID of type idsLdapFilterProperties (string).

false

iplanetFiltersRef
Specifies the list of default IPlanet LDAP filters.

Configuration ID of type iplanetLdapFilterProperties (string).

false

netscapeFiltersRef
Specifies the list of default Netscape LDAP filters.

Configuration ID of type netscapeLdapFilterProperties (string).

false

securewayFiltersRef
Specifies the list of default SecureWay LDAP filters.

Configuration ID of type securewayLdapFilterProperties (string).

false

failoverServers
List of LDAP failover servers.

List of LDAP failover servers.

name
failoverServers.name.desc

string

false

server
Configuration properties for LDAP failover server.

Configuration properties for LDAP failover server.

host
LDAP server host name, which can be either an IP address or a domain name service (DNS) name.

string

true

port
LDAP failover server port.

int

true

false

false

activedFilters
    Specifies the list of default Active Directory LDAP filters.

    Element of type activedLdapFilterProperties.

    false

customFilters
    Specifies the list of default Custom LDAP filters.

    Element of type customLdapFilterProperties.

    false

domino50Filters
    Specifies the list of default Domino LDAP filters.

    Element of type domino50LdapFilterProperties.

    false

edirectoryFilters
    Specifies the list of EDirectory LDAP filters.

    Element of type edirectoryLdapFilterProperties.

    false

idsFilters
    Specifies the list of default IBM Directory Server LDAP filters.

    Element of type idsLdapFilterProperties.

    false

iplanetFilters
    Specifies the list of default IPlanet LDAP filters.

    Element of type iplanetLdapFilterProperties.

    false

netscapeFilters
    Specifies the list of default Netscape LDAP filters.

    Element of type netscapeLdapFilterProperties.

    false

securewayFilters
    Specifies the list of default SecureWay LDAP filters.

    Element of type securewayLdapFilterProperties.

    false

library

Shared Library. PID is com.ibm.ws.classloading.sharedlibrary.

name
    Name of shared library for administrators

    string

    false

description
> Description of shared library for administrators
>
> string
>
> false

filesetRef
> Id of referenced Fileset
>
> List of configuration IDs of type fileset (comma-separated string).
>
> false

apiTypeVisibility
> The types of API package this library's class loader will be able to see, as a comma-separated list of any combination of the following: spec, ibm-api, ibm-spi, third-party.
>
> string
>
> spec,ibm-api
>
> false

fileset
> Id of referenced Fileset
>
> Element of type fileset.
>
> false

logging

Controls the output of log and trace messages. PID is com.ibm.ws.logging.

maxFileSize
> Maximum size of a log file, in megabytes, before being rolled over; a value of 0 means no limit.
>
> com.ibm.ws.logging.max.file.size
>
> int
>
> 20
>
> false

maxFiles
> Maximum number of log files that will be kept, before the oldest file is removed; a value of 0 means no limit.
>
> com.ibm.ws.logging.max.files
>
> int
>
> 2
>
> false

logDirectory
> Location to which all log files will be written. The default value is ${server.output.dir}/logs.
>
> com.ibm.ws.logging.log.directory
>
> string
>
> ${server.output.dir}/logs
>
> false

messageFileName

Name of the file to which message output will be written relative to the configured log directory. The default value is messages.log.

com.ibm.ws.logging.message.file.name

string

messages.log

false

traceFileName

Name of the file to which trace output will be written relative to the configured log directory. The default value is trace.log.

com.ibm.ws.logging.trace.file.name

string

trace.log

false

traceSpecification

A trace specification that conforms to the trace specification grammar and specifies the initial state for various trace components. An empty value is allowed and treated as 'disable all trace'. Any component that is not specified is initialized to a default state of disabled.

com.ibm.ws.logging.trace.specification

string

*=info=enabled

false

traceFormat

This format is used for the trace log.

com.ibm.ws.logging.trace.format

string

ENHANCED

BASIC

Use the basic trace format.

ENHANCED

Use the enhanced basic trace format.

ADVANCED

Use the advanced trace format.

false

consoleLogLevel

The logging level used to filter messages written to system streams. The default value is audit.

com.ibm.ws.logging.console.log.level

string

AUDIT

INFO Info, audit, and warning messages will be written to the system output stream. Error messages will be written to the system error stream.

AUDIT

Audit and warning messages will be written to the system output stream. Error messages will be written to the system error stream.

WARNING

Warning messages will be written to the system output stream. Error messages will be written to the system error stream.

ERROR

Error messages will be written to the system error stream.

OFF

false

ltpa

Lightweight Third Party Authentication (LTPA) token configuration. PID is com.ibm.ws.security.token.ltpa.LTPAConfiguration.

keysFileName

Path of the file containing the token keys.

string

${server.config.dir}/resources/security/ltpa.keys

false

keysPassword

Password for the token keys. The value can be stored in clear text or encoded form. It is recommended to encode the password, use the securityUtility tool with the encode option.

password (string)

{xor}CDo9Hgw=

false

expiration

Amount of time after which a token expires in minutes.

long

120

false

mimeTypes

Definition of mime types shared by all http virtual hosts. PID is com.ibm.ws.http.mimetype.

type

Definition of mime type as id=value. Use the extension as the id, and the associated type as the value.

string

false

monitor

Configuration for Monitoring. PID is com.ibm.ws.monitor.internal.MonitoringFrameworkExtender.

enableTraditionalPMI

Internal Property to enable or disable Traditional PMI

boolean

false

false

nativeTransactionManager

Configures the RRS Native Transaction Manager. PID is com.ibm.ws.zos.tx.

shutdownTimeout
> The amount of time to wait before allowing the native context manager to stop if one or more native contexts are in-use. A native context is considered in-use if it contains an RRS unit of recovery with one or more interests. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
>
> string
>
> 15s
>
> false

resourceManagerNamePrefix
> The resource manager prefix to be used as part of the server generated resource manager name.
>
> string
>
> DEFAULT
>
> false

netscapeLdapFilterProperties

Specifies the list of default Netscape LDAP filters. PID is com.ibm.ws.security.registry.ldap.internal.filters.netscape.

userFilter
> An LDAP filter clause for searching the user registry for users.
>
> string
>
> (&(uid=%v)(objectclass=inetOrgPerson))
>
> true

groupFilter
> An LDAP filter clause for search the user registry for groups.
>
> string
>
> (&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))
>
> true

userIdMap
> An LDAP filter that maps the name of a user to an LDAP entry.
>
> string
>
> inetOrgPerson:uid
>
> true

groupIdMap
> groupId.desc
>
> string
>
> *:cn

true

groupMemberIdMap
An LDAP filter that identifies user to group memberships.

string

groupOfNames:member;groupOfUniqueNames:uniqueMember

true

pluginConfiguration
Generate plugin configuration. PID is com.ibm.ws.generatePluginConfig.

pluginInstallRoot
Web container plugin installation position in file system

string

.

false

webserverPort
Web server HTTP port

string

80

false

webserverSecurePort
Web server HTTPS port

string

443

false

sslKeyringLocation
Location of SSL keyring

string

PATH/FILE

false

sslStashfileLocation
Location of SSL stashfile

string

PATH/FILE

false

sslCertlabel
SSL cert label

string

REPLACE

false

properties

List of JDBC vendor properties for the data source. For example, databaseName="dbname" serverName="localhost" portNumber="50000". PID is com.ibm.ws.jdbc.dataSource.properties, and it is the child of complex type "dataSource".

databaseName
    JDBC driver property: databaseName.

    string

    false

serverName
    Server where the database is running.

    string

    false

portNumber
    Port on which to obtain database connections.

    int

    false

URL
    URL for connecting to the database.

    string

    false

user
    Database user name.

    string

    false

password
    Password for database user.

    password (string)

    false

properties.datadirect.sqlserver

Data source properties for the DataDirect Connect for JDBC driver for Microsoft SQL Server. PID is com.ibm.ws.jdbc.dataSource.properties.datadirect.sqlserver, and it is the child of complex type "dataSource".

databaseName
    JDBC driver property: databaseName.

    string

    false

serverName
    Server where the database is running.

    string

    localhost

    false

portNumber
    Port on which to obtain database connections.

    int

    1433

    false

user
    Database user name.

    string

    false

password
    Password for database user.

    password (string)

    false

accountingInfo
    JDBC driver property: accountingInfo.

    string

    false

alternateServers
    JDBC driver property: alternateServers.

    string

    false

alwaysReportTriggerResults
    JDBC driver property: alwaysReportTriggerResults.

    boolean

    false

applicationName
    JDBC driver property: applicationName.

    string

    false

authenticationMethod
    JDBC driver property: authenticationMethod.

    string

    auto

    kerberos

    ntlm

    userIdPassword

    false

bulkLoadBatchSize
    JDBC driver property: bulkLoadBatchSize.

    long

false

bulkLoadOptions
JDBC driver property: bulkLoadOptions.

long

false

clientHostName
JDBC driver property: clientHostName.

string

false

clientUser
JDBC driver property: clientUser.

string

false

codePageOverride
JDBC driver property: codePageOverride.

string

false

connectionRetryCount
JDBC driver property: connectionRetryCount.

int

false

connectionRetryDelay
JDBC driver property: connectionRetryDelay. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

convertNull
JDBC driver property: convertNull.

int

false

dateTimeInputParameterType
JDBC driver property: dateTimeInputParameterType.

string

auto

dateTime

dateTimeOffset

false

dateTimeOutputParameterType
JDBC driver property: dateTimeOutputParameterType.

string

> **auto**
>
> **dateTime**
>
> **dateTimeOffset**
>
> false

describeInputParameters
JDBC driver property: describeInputParameters.

> string
>
> **describeAll**
>
> **describeIfDateTime**
>
> **describeIfString**
>
> **noDescribe**
>
> false

describeOutputParameters
JDBC driver property: describeOutputParameters.

> string
>
> **describeAll**
>
> **describeIfDateTime**
>
> **describeIfString**
>
> **noDescribe**
>
> false

enableBulkLoad
JDBC driver property: enableBulkLoad.

> boolean
>
> false

enableCancelTimeout
JDBC driver property: enableCancelTimeout.

> boolean
>
> false

encryptionMethod
JDBC driver property: encryptionMethod.

> string
>
> **noEncryption**
>
> **loginSSL**
>
> **requestSSL**
>
> **SSL**
>
> false

failoverGranularity
JDBC driver property: failoverGranularity.

> string

atomic

atomicWithRepositioning

disableIntegrityCheck

nonAtomic

false

`failoverMode`
JDBC driver property: failoverMode.

string

connect

extended

select

false

`failoverPreconnect`
JDBC driver property: failoverPreconnect.

boolean

false

`hostNameInCertificate`
JDBC driver property: hostNameInCertificate.

string

false

`initializationString`
JDBC driver property: initializationString.

string

false

`insensitiveResultSetBufferSize`
JDBC driver property: insensitiveResultSetBufferSize.

int

false

`javaDoubleToString`
JDBC driver property: javaDoubleToString.

boolean

false

`JDBCBehavior`
JDBC driver property: JDBCBehavior. Values are: 0 (JDBC 4.0) or 1 (JDBC 3.0).

int

0

0   JDBC 4.0

1   JDBC 3.0

false

loadBalancing
JDBC driver property: loadBalancing.

boolean

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

longDataCacheSize
JDBC driver property: longDataCacheSize.

int

false

netAddress
JDBC driver property: netAddress.

string

false

packetSize
JDBC driver property: packetSize.

int

false

queryTimeout
queryTimeout.datadirect.sqlserver.desc. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

resultsetMetaDataOptions
JDBC driver property: resultsetMetaDataOptions.

int

false

selectMethod
JDBC driver property: selectMethod.

string

cursor

direct

false

snapshotSerializable
JDBC driver property: snapshotSerializable.

boolean

false

spyAttributes
    JDBC driver property: spyAttributes.

    string

    false

stringInputParameterType
    JDBC driver property: stringInputParameterType.

    string

    varchar

    nvarchar

    varchar

    false

stringOutputParameterType
    JDBC driver property: stringOutputParameterType.

    string

    varchar

    nvarchar

    varchar

    false

suppressConnectionWarnings
    JDBC driver property: suppressConnectionWarnings.

    boolean

    false

transactionMode
    JDBC driver property: transactionMode.

    string

    explicit

    implicit

    false

truncateFractionalSeconds
    JDBC driver property: truncateFractionalSeconds.

    boolean

    false

trustStore
    JDBC driver property: trustStore.

    string

    false

trustStorePassword
    JDBC driver property: trustStorePassword.

    password (string)

    false

useServerSideUpdatableCursors
>   JDBC driver property: useServerSideUpdatableCursors.

>   boolean

>   false

validateServerCertificate
>   JDBC driver property: validateServerCertificate.

>   boolean

>   false

XATransactionGroup
>   JDBC driver property: XATransactionGroup.

>   string

>   false

XMLDescribeType
>   JDBC driver property: XMLDescribeType.

>   string

>   **longvarbinary**

>   **longvarchar**

>   false

properties.db2.i.native

Data source properties for the IBM DB2 for i Native JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.db2.i.native, and it is the child of complex type "dataSource".

databaseName
>   JDBC driver property: databaseName.

>   string

>   *LOCAL

>   false

user
>   Database user name.

>   string

>   false

password
>   Password for database user.

>   password (string)

>   false

access
>   JDBC driver property: access.

>   string

>   all

>   **all**

>   **read call**

read only

false

`autoCommit`
   JDBC driver property: autoCommit.

   boolean

   true

   false

`batchStyle`
   JDBC driver property: batchStyle.

   string

   2.0

   2.0

   2.1

   false

`behaviorOverride`
   JDBC driver property: behaviorOverride.

   int

   false

`blockSize`
   JDBC driver property: blockSize.

   int

   32

   0

   8

   16

   32

   64

   128

   256

   512

   false

`cursorHold`
   JDBC driver property: cursorHold.

   boolean

   false

   false

`cursorSensitivity`
   JDBC driver property: cursorSensitivity. Values are: 0 (TYPE_SCROLL_SENSITIVE_STATIC), 1
   (TYPE_SCROLL_SENSITIVE_DYNAMIC), 2 (TYPE_SCROLL_ASENSITIVE).

   string

asensitive

asensitive

sensitive

false

dataTruncation
JDBC driver property: dataTruncation.

string

true

false

dateFormat
JDBC driver property: dateFormat.

string

dmy

eur

mdy

iso

jis

julian

usa

ymd

false

dateSeparator
JDBC driver property: dateSeparator.

string

/   The forward slash character (/).

-   The dash character (-).

.   The period character (.).

,   The comma character (,).

b   The character b

false

decimalSeparator
JDBC driver property: decimalSeparator.

string

.   The period character (.).

,   The comma character (,).

false

directMap
JDBC driver property: directMap.

boolean

false

false

doEscapeProcessing
JDBC driver property: doEscapeProcessing.

boolean

true

false

fullErrors
JDBC driver property: fullErrors.

boolean

false

libraries
JDBC driver property: libraries.

string

false

lobThreshold
JDBC driver property: lobThreshold.

int

0

false

lockTimeout
JDBC driver property: lockTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

0

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

maximumPrecision
JDBC driver property: maximumPrecision.

int

31

31

63

false

maximumScale
: JDBC driver property: maximumScale.

    int

    31

    false

minimumDivideScale
: JDBC driver property: minimumDivideScale.

    int

    0

    false

networkProtocol
: JDBC driver property: networkProtocol.

    int

    false

portNumber
: Port on which to obtain database connections.

    int

    false

prefetch
: JDBC driver property: prefetch.

    boolean

    true

    false

queryOptimizeGoal
: JDBC driver property: queryOptimizeGoal. Values are: 1 (*FIRSTIO) or 2 (*ALLIO).

    string

    2

    1   *FIRSTIO

    2   *ALLIO

    false

reuseObjects
: JDBC driver property: reuseObjects.

    boolean

    true

    false

serverName
: Server where the database is running.

    string

    false

serverTraceCategories

JDBC driver property: serverTraceCategories.

int

0

false

systemNaming

JDBC driver property: systemNaming.

boolean

false

false

timeFormat

JDBC driver property: timeFormat.

string

eur

hms

iso

jis

usa

false

timeSeparator

JDBC driver property: timeSeparator.

string

:   The colon character (:).

.   The period character (.).

,   The comma character (,).

b   The character b

false

trace

JDBC driver property: trace.

boolean

false

transactionTimeout

JDBC driver property: transactionTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

0

false

translateBinary

JDBC driver property: translateBinary.

boolean

false

false

translateHex
JDBC driver property: translateHex.

string

character

binary

character

false

useBlockInsert
JDBC driver property: useBlockInsert.

boolean

false

false

properties.db2.i.toolbox

Data source properties for the IBM DB2 for i Toolbox JDBC driver. PID is
com.ibm.ws.jdbc.dataSource.properties.db2.i.toolbox, and it is the child of complex type "dataSource".

serverName
Server where the database is running.

string

true

databaseName
JDBC driver property: databaseName.

string

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

access
JDBC driver property: access.

string

all

all

read call

read only

false

`behaviorOverride`
JDBC driver property: behaviorOverride.

int

false

`bidiImplicitReordering`
JDBC driver property: bidiImplicitReordering.

boolean

true

false

`bidiNumericOrdering`
JDBC driver property: bidiNumericOrdering.

boolean

false

false

`bidiStringType`
JDBC driver property: bidiStringType.

int

false

`bigDecimal`
JDBC driver property: bigDecimal.

boolean

true

false

`blockCriteria`
JDBC driver property: blockCriteria. Values are: 0 (no record blocking), 1 (block if FOR FETCH ONLY is specified), 2 (block if FOR UPDATE is specified).

int

2

0

1

2

false

`blockSize`
JDBC driver property: blockSize.

int

32

0

8

16

32

64

128

256

512

false

cursorHold
JDBC driver property: cursorHold.

boolean

false

false

cursorSensitivity
JDBC driver property: cursorSensitivity. Values are: 0 (TYPE_SCROLL_SENSITIVE_STATIC), 1 (TYPE_SCROLL_SENSITIVE_DYNAMIC), 2 (TYPE_SCROLL_ASENSITIVE).

string

asensitive

asensitive

insensitive

sensitive

false

dataCompression
JDBC driver property: dataCompression.

boolean

true

false

dataTruncation
JDBC driver property: dataTruncation.

boolean

true

false

dateFormat
JDBC driver property: dateFormat.

string

dmy

eur

mdy

iso

jis

> **julian**
>
> **usa**
>
> **ymd**
>
> false

dateSeparator
> JDBC driver property: dateSeparator.
>
> string
>
> **/**   The forward slash character (/).
>
> **-**   The dash character (-).
>
> **.**   The period character (.).
>
> **,**   The comma character (,).
>
>   The space character ( ).
>
> false

decimalSeparator
> JDBC driver property: decimalSeparator.
>
> string
>
> **.**   The period character (.).
>
> **,**   The comma character (,).
>
> false

driver
> JDBC driver property: driver.
>
> string
>
> toolbox
>
> **native**
>
> **toolbox**
>
> false

errors
> JDBC driver property: errors.
>
> string
>
> basic
>
> **basic**
>
> **full**
>
> false

extendedDynamic
> JDBC driver property: extendedDynamic.
>
> boolean
>
> false
>
> false

extendedMetaData
　　JDBC driver property: extendedMetaData.

　　boolean

　　false

　　false

fullOpen
　　JDBC driver property: fullOpen.

　　boolean

　　false

　　false

holdInputLocators
　　JDBC driver property: holdInputLocators.

　　boolean

　　true

　　false

holdStatements
　　JDBC driver property: holdStatements.

　　boolean

　　false

　　false

isolationLevelSwitchingSupport
　　JDBC driver property: isolationLevelSwitchingSupport.

　　boolean

　　false

　　false

keepAlive
　　JDBC driver property: keepAlive.

　　boolean

　　false

lazyClose
　　JDBC driver property: lazyClose.

　　boolean

　　false

　　false

libraries
　　JDBC driver property: libraries.

　　string

　　false

lobThreshold
　　JDBC driver property: lobThreshold.

int

0

false

loginTimeout

JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

maximumPrecision

JDBC driver property: maximumPrecision.

int

31

31

63   64

false

maximumScale

JDBC driver property: maximumScale.

int

31

false

metaDataSource

JDBC driver property: metaDataSource.

int

1

false

minimumDivideScale

JDBC driver property: minimumDivideScale.

int

0

false

naming

JDBC driver property: naming.

string

sql

sql

system

false

package

JDBC driver property: package.

string

false

packageAdd
JDBC driver property: packageAdd.

boolean

true

false

packageCCSID
JDBC driver property: packageCCSID. Values are: 1200 (UCS-2) or 13488 (UTF-16).

int

13488

1200   1200 (UCS-2)

13488   13488 (UTF-16)

false

packageCache
JDBC driver property: packageCache.

boolean

false

false

packageCriteria
JDBC driver property: packageCriteria.

string

default

default

select

false

packageError
JDBC driver property: packageError.

string

warning

exception

warning

none

false

packageLibrary
JDBC driver property: packageLibrary.

string

QGPL

false

prefetch
    JDBC driver property: prefetch.

    boolean

    true

    false

prompt
    JDBC driver property: prompt.

    boolean

    false

    false

proxyServer
    JDBC driver property: proxyServer.

    string

    false

qaqqiniLibrary
    JDBC driver property: qaqqiniLibrary.

    string

    false

queryOptimizeGoal
    JDBC driver property: queryOptimizeGoal. Values are: 1 (*FIRSTIO) or 2 (*ALLIO).

    int

    0

    false

receiveBufferSize
    JDBC driver property: receiveBufferSize.

    int

    false

remarks
    JDBC driver property: remarks.

    string

    system

    sql

    system

    false

rollbackCursorHold
    JDBC driver property: rollbackCursorHold.

    boolean

    false

    false

savePasswordWhenSerialized
    JDBC driver property: savePasswordWhenSerialized.

    boolean

    false

    false

secondaryUrl
    JDBC driver property: secondaryUrl.

    string

    false

secure
    JDBC driver property: secure.

    boolean

    false

    false

sendBufferSize
    JDBC driver property: sendBufferSize.

    int

    false

serverTraceCategories
    JDBC driver property: serverTraceCategories.

    int

    0

    false

soLinger
    JDBC driver property: soLinger. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

    string

    false

soTimeout
    JDBC driver property: soTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

    string

    false

sort
    JDBC driver property: sort.

    string

    hex

    hex

    language

table

false

sortLanguage
   JDBC driver property: sortLanguage.

   string

   false

sortTable
   JDBC driver property: sortTable.

   string

   false

sortWeight
   JDBC driver property: sortWeight.

   string

   shared

   unqiue
            unique

   false

tcpNoDelay
   JDBC driver property: tcpNoDelay.

   boolean

   false

threadUsed
   JDBC driver property: threadUsed.

   boolean

   true

   false

timeFormat
   JDBC driver property: timeFormat.

   string

   eur

   hms

   iso

   jis

   usa

   false

timeSeparator
   JDBC driver property: timeSeparator.

   string

   :   The colon character (:).

   .   The period character (.).

, The comma character (,).

The space character ( ).

false

toolboxTrace
JDBC driver property: toolboxTrace.

string

none

datastream

diagnostic

error

information

warning

conversion

proxy

pcml

jdbc

all

thread

false

trace
JDBC driver property: trace.

boolean

false

translateBinary
JDBC driver property: translateBinary.

boolean

false

false

translateBoolean
JDBC driver property: translateBoolean.

boolean

true

false

translateHex
JDBC driver property: translateHex.

string

character

binary

character

false

trueAutoCommit
    JDBC driver property: trueAutoCommit.

    boolean

    false

    false

xaLooselyCoupledSupport
    JDBC driver property: xaLooselyCoupledSupport.

    int

    0

    false

properties.db2.jcc

Data source properties for the IBM Data Server Driver for JDBC and SQLJ for DB2. PID is com.ibm.ws.jdbc.dataSource.properties.db2.jcc, and it is the child of complex type "dataSource".

driverType
    JDBC driver property: driverType.

    int

    4

    2   Type 2 JDBC driver.

    4   Type 4 JDBC driver.

    false

databaseName
    JDBC driver property: databaseName.

    string

    false

serverName
    Server where the database is running.

    string

    localhost

    false

portNumber
    Port on which to obtain database connections.

    int

    50000

    false

user
    Database user name.

    string

    false

password
> Password for database user.

> password (string)

> false

blockingReadConnectionTimeout
> JDBC driver property: blockingReadConnectionTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

> string

> false

clientAccountingInformation
> JDBC driver property: clientAccountingInformation.

> string

> false

clientApplicationInformation
> JDBC driver property: clientApplicationInformation.

> string

> false

clientRerouteServerListJNDIName
> JDBC driver property: clientRerouteServerListJNDIName.

> string

> false

clientUser
> JDBC driver property: clientUser.

> string

> false

clientWorkstation
> JDBC driver property: clientWorkstation.

> string

> false

currentFunctionPath
> JDBC driver property: currentFunctionPath.

> string

> false

currentLockTimeout
> JDBC driver property: currentLockTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

> string

> false

currentPackagePath
> JDBC driver property: currentPackagePath.

string

false

currentPackageSet
  JDBC driver property: currentPackageSet.

  string

  false

currentSQLID
  JDBC driver property: currentSQLID.

  string

  false

currentSchema
  JDBC driver property: currentSchema.

  string

  false

cursorSensitivity
  JDBC driver property: cursorSensitivity. Values are: 0 (TYPE_SCROLL_SENSITIVE_STATIC), 1
  (TYPE_SCROLL_SENSITIVE_DYNAMIC), 2 (TYPE_SCROLL_ASENSITIVE).

  int

  0   TYPE_SCROLL_SENSITIVE_STATIC

  1   TYPE_SCROLL_SENSITIVE_DYNAMIC

  2   TYPE_SCROLL_ASENSITIVE

  false

deferPrepares
  JDBC driver property: deferPrepares.

  boolean

  true

  false

enableClientAffinitiesList
  JDBC driver property: enableClientAffinitiesList. Values are: 1 (YES) or 2 (NO).

  int

  1   YES

  2   NO

  false

enableNamedParameterMarkers
  JDBC driver property: enableNamedParameterMarkers. Values are: 1 (YES) or 2 (NO).

  int

  1   YES

  2   NO

  false

enableSeamlessFailover

JDBC driver property: enableSeamlessFailover. Values are: 1 (YES) or 2 (NO).

int

1 YES

2 NO

false

enableSysplexWLB

JDBC driver property: enableSysplexWLB.

boolean

false

fetchSize

JDBC driver property: fetchSize.

int

false

fullyMaterializeInputStreams

JDBC driver property: fullyMaterializeInputStreams.

boolean

false

fullyMaterializeLobData

JDBC driver property: fullyMaterializeLobData.

boolean

false

keepDynamic

JDBC driver property: keepDynamic.

int

false

kerberosServerPrincipal

JDBC driver property: kerberosServerPrincipal.

string

false

loginTimeout

JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

maxRetriesForClientReroute

JDBC driver property: maxRetriesForClientReroute.

int

false

queryCloseImplicit

JDBC driver property: queryCloseImplicit. Values are: 1 (QUERY_CLOSE_IMPLICIT_YES) or 2 (QUERY_CLOSE_IMPLICIT_NO).

int

1   QUERY_CLOSE_IMPLICIT_YES

2   QUERY_CLOSE_IMPLICIT_NO

false

queryDataSize

JDBC driver property: queryDataSize.

int

false

readOnly

JDBC driver property: readOnly.

boolean

false

resultSetHoldability

JDBC driver property: resultSetHoldability. Values are: 1 (HOLD_CURSORS_OVER_COMMIT) or 2 (CLOSE_CURSORS_AT_COMMIT).

int

1   HOLD_CURSORS_OVER_COMMIT

2   CLOSE_CURSORS_AT_COMMIT

false

resultSetHoldabilityForCatalogQueries

JDBC driver property: resultSetHoldabilityForCatalogQueries. Values are: 1 (HOLD_CURSORS_OVER_COMMIT) or 2 (CLOSE_CURSORS_AT_COMMIT).

int

1   HOLD_CURSORS_OVER_COMMIT

2   CLOSE_CURSORS_AT_COMMIT

false

retrieveMessagesFromServerOnGetMessage

JDBC driver property: retrieveMessagesFromServerOnGetMessage.

boolean

true

false

retryIntervalForClientReroute

JDBC driver property: retryIntervalForClientReroute. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

securityMechanism

JDBC driver property: securityMechanism. Values are: 3 (CLEAR_TEXT_PASSWORD_SECURITY), 4

(USER_ONLY_SECURITY), 7 (ENCRYPTED_PASSWORD_SECURITY), 9
(ENCRYPTED_USER_AND_PASSWORD_SECURITY), 11 (KERBEROS_SECURITY), 12
(ENCRYPTED_USER_AND_DATA_SECURITY"),
(ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY"), 15 (PLUGIN_SECURITY), 16
(ENCRYPTED_USER_ONLY_SECURITY).

int

3	CLEAR_TEXT_PASSWORD_SECURITY
4	USER_ONLY_SECURITY
7	ENCRYPTED_PASSWORD_SECURITY
9	ENCRYPTED_USER_AND_PASSWORD_SECURITY
11	KERBEROS_SECURITY
12	ENCRYPTED_USER_AND_DATA_SECURITY
13	ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY
15	PLUGIN_SECURITY
16	ENCRYPTED_USER_ONLY_SECURITY

false

sendDataAsIs
JDBC driver property: sendDataAsIs.

boolean

false

sqljEnableClassLoaderSpecificProfiles
JDBC driver property: sqljEnableClassLoaderSpecificProfiles.

boolean

false

sslConnection
JDBC driver property: sslConnection.

boolean

false

streamBufferSize
JDBC driver property: streamBufferSize.

int

false

sysSchema
JDBC driver property: sysSchema.

string

false

traceDirectory
JDBC driver property: traceDirectory.

string

false

traceFile
JDBC driver property: traceFile.

string

false

traceFileAppend
JDBC driver property: traceFileAppend.

boolean

false

traceLevel
Bitwise combination of the following constant values: TRACE_NONE=0,
TRACE_CONNECTION_CALLS=1, TRACE_STATEMENT_CALLS=2,
TRACE_RESULT_SET_CALLS=4, TRACE_DRIVER_CONFIGURATION=16, TRACE_CONNECTS=32,
TRACE_DRDA_FLOWS=64, TRACE_RESULT_SET_META_DATA=128,
TRACE_PARAMETER_META_DATA=256, TRACE_DIAGNOSTICS=512, TRACE_SQLJ=1024,
TRACE_META_CALLS=8192, TRACE_DATASOURCE_CALLS=16384,
TRACE_LARGE_OBJECT_CALLS=32768, TRACE_SYSTEM_MONITOR=131072,
TRACE_TRACEPOINTS=262144, TRACE_ALL=-1.

int

0

false

useCachedCursor
JDBC driver property: useCachedCursor.

boolean

false

useJDBC4ColumnNameAndLabelSemantics
JDBC driver property: useJDBC4ColumnNameAndLabelSemantics. Values are: 1 (YES) or 2 (NO).

int

1   YES

2   NO

false

useTransactionRedirect
JDBC driver property: useTransactionRedirect.

boolean

false

xaNetworkOptimization
JDBC driver property: xaNetworkOptimization.

boolean

false

properties.derby.client

Data source properties for Derby Network Client JDBC driver. PID is
com.ibm.ws.jdbc.dataSource.properties.derby.client, and it is the child of complex type "dataSource".

createDatabase
    JDBC driver property: createDatabase.

    string

    create When the first connection is established, automatically create the database if it doesn't exist.

    false   Do not automatically create the database.

    false

databaseName
    JDBC driver property: databaseName.

    string

    false

serverName
    Server where the database is running.

    string

    localhost

    false

portNumber
    Port on which to obtain database connections.

    int

    1527

    false

user
    Database user name.

    string

    false

password
    Password for database user.

    password (string)

    false

connectionAttributes
    JDBC driver property: connectionAttributes.

    string

    false

loginTimeout
    JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

    string

    false

retrieveMessageText
    JDBC driver property: retrieveMessageText.

    boolean

true

false

securityMechanism

JDBC driver property: securityMechanism. Values are: 3 (CLEAR_TEXT_PASSWORD_SECURITY), 4 (USER_ONLY_SECURITY), 7 (ENCRYPTED_PASSWORD_SECURITY), 8 (STRONG_PASSWORD_SUBSTITUTE_SECURITY), 9 (ENCRYPTED_USER_AND_PASSWORD_SECURITY).

short

3

3	CLEAR_TEXT_PASSWORD_SECURITY
4	USER_ONLY_SECURITY
7	ENCRYPTED_PASSWORD_SECURITY
8	STRONG_PASSWORD_SUBSTITUTE_SECURITY
9	ENCRYPTED_USER_AND_PASSWORD_SECURITY

false

shutdownDatabase

JDBC driver property: shutdownDatabase.

string

shutdown	Shut down the database when a connection is attempted.
false	Do not shut down the database.

false

ssl

JDBC driver property: ssl.

string

basic

peerAuthentication

off

false

traceDirectory

JDBC driver property: traceDirectory.

string

false

traceFile

JDBC driver property: traceFile.

string

false

traceFileAppend

JDBC driver property: traceFileAppend.

boolean

false

traceLevel
   Bitwise combination of the following constant values: TRACE_NONE=0,
   TRACE_CONNECTION_CALLS=1, TRACE_STATEMENT_CALLS=2,
   TRACE_RESULT_SET_CALLS=4, TRACE_DRIVER_CONFIGURATION=16, TRACE_CONNECTS=32,
   TRACE_DRDA_FLOWS=64, TRACE_RESULT_SET_META_DATA=128,
   TRACE_PARAMETER_META_DATA=256, TRACE_DIAGNOSTICS=512, TRACE_XA_CALLS=2048,
   TRACE_ALL=-1.

   int

   false

properties.derby.embedded

Data source properties for Derby Embedded JDBC driver. PID is
com.ibm.ws.jdbc.dataSource.properties.derby.embedded, and it is the child of complex type "dataSource".

createDatabase
   JDBC driver property: createDatabase.

   string

   create When the first connection is established, automatically create the database if it doesn't exist.

   false Do not automatically create the database.

   false

databaseName
   JDBC driver property: databaseName.

   string

   false

user
   Database user name.

   string

   false

password
   Password for database user.

   password (string)

   false

connectionAttributes
   JDBC driver property: connectionAttributes.

   string

   false

loginTimeout
   JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be
   hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include
   multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

   string

   false

shutdownDatabase
   JDBC driver property: shutdownDatabase.

string

shutdown
> Shut down the database when a connection is attempted.

false   Do not shut down the database.

false

properties.informix

Data source properties for the Informix® JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.informix, and it is the child of complex type "dataSource".

`databaseName`
> JDBC driver property: databaseName.

> string

> false

`ifxIFXHOST`
> JDBC driver property: ifxIFXHOST.

> string

> localhost

> false

`serverName`
> Server where the database is running.

> string

> false

`portNumber`
> Port on which to obtain database connections.

> int

> 1526

> false

`user`
> Database user name.

> string

> false

`password`
> Password for database user.

> password (string)

> false

`ifxCLIENT_LOCALE`
> JDBC driver property: ifxCLIENT_LOCALE.

> string

> false

`ifxCPMAgeLimit`
> JDBC driver property: ifxCPMAgeLimit. Specify a positive integer followed by a unit of time, which can

be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

ifxCPMInitPoolSize
JDBC driver property: ifxCPMInitPoolSize.

int

false

ifxCPMMaxConnections
JDBC driver property: ifxCPMMaxConnections.

int

false

ifxCPMMaxPoolSize
JDBC driver property: ifxCPMMaxPoolSize.

int

false

ifxCPMMinAgeLimit
JDBC driver property: ifxCPMMinAgeLimit. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

ifxCPMMinPoolSize
JDBC driver property: ifxCPMMinPoolSize.

int

false

ifxCPMServiceInterval
JDBC driver property: ifxCPMServiceInterval. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

false

ifxDBANSIWARN
JDBC driver property: ifxDBANSIWARN.

boolean

false

ifxDBCENTURY
JDBC driver property: ifxDBCENTURY.

string

false

ifxDBDATE
    JDBC driver property: ifxDBDATE.

    string

    false

ifxDBSPACETEMP
    JDBC driver property: ifxDBSPACETEMP.

    string

    false

ifxDBTEMP
    JDBC driver property: ifxDBTEMP.

    string

    false

ifxDBTIME
    JDBC driver property: ifxDBTIME.

    string

    false

ifxDBUPSPACE
    JDBC driver property: ifxDBUPSPACE.

    string

    false

ifxDB_LOCALE
    JDBC driver property: ifxDB_LOCALE.

    string

    false

ifxDELIMIDENT
    JDBC driver property: ifxDELIMIDENT.

    boolean

    false

ifxENABLE_TYPE_CACHE
    JDBC driver property: ifxENABLE_TYPE_CACHE.

    boolean

    false

ifxFET_BUF_SIZE
    JDBC driver property: ifxFET_BUF_SIZE.

    int

    false

ifxGL_DATE
    JDBC driver property: ifxGL_DATE.

    string

    false

ifxGL_DATETIME

JDBC driver property: ifxGL_DATETIME.

string

false

ifxIFX_AUTOFREE

JDBC driver property: ifxIFX_AUTOFREE.

boolean

false

ifxIFX_DIRECTIVES

JDBC driver property: ifxIFX_DIRECTIVES.

string

false

ifxIFX_LOCK_MODE_WAIT

ifxIFX_LOCK_MODE_WAIT.desc. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

2s

false

ifxIFX_SOC_TIMEOUT

JDBC driver property: ifxIFX_SOC_TIMEOUT. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

string

false

ifxIFX_USEPUT

JDBC driver property: ifxIFX_USEPUT.

boolean

false

ifxIFX_USE_STRENC

JDBC driver property: ifxIFX_USE_STRENC.

boolean

false

ifxIFX_XASPEC

JDBC driver property: ifxIFX_XASPEC.

string

y

false

ifxINFORMIXCONRETRY

JDBC driver property: ifxINFORMIXCONRETRY.

int

false

ifxINFORMIXCONTIME
>   JDBC driver property: ifxINFORMIXCONTIME. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.
>
>   string
>
>   false

ifxINFORMIXOPCACHE
>   JDBC driver property: ifxINFORMIXOPCACHE.
>
>   string
>
>   false

ifxINFORMIXSTACKSIZE
>   JDBC driver property: ifxINFORMIXSTACKSIZE.
>
>   int
>
>   false

ifxJDBCTEMP
>   JDBC driver property: ifxJDBCTEMP.
>
>   string
>
>   false

ifxLDAP_IFXBASE
>   JDBC driver property: ifxLDAP_IFXBASE.
>
>   string
>
>   false

ifxLDAP_PASSWD
>   JDBC driver property: ifxLDAP_PASSWD.
>
>   string
>
>   false

ifxLDAP_URL
>   JDBC driver property: ifxLDAP_URL.
>
>   string
>
>   false

ifxLDAP_USER
>   JDBC driver property: ifxLDAP_USER.
>
>   string
>
>   false

ifxLOBCACHE
>   JDBC driver property: ifxLOBCACHE.
>
>   int
>
>   false

ifxNEWCODESET
>   JDBC driver property: ifxNEWCODESET.

string

false

ifxNEWLOCALE
>JDBC driver property: ifxNEWLOCALE.

>string

>false

ifxNODEFDAC
>JDBC driver property: ifxNODEFDAC.

>string

>false

ifxOPTCOMPIND
>JDBC driver property: ifxOPTCOMPIND.

>string

>false

ifxOPTOFC
>JDBC driver property: ifxOPTOFC.

>string

>false

ifxOPT_GOAL
>JDBC driver property: ifxOPT_GOAL.

>string

>false

ifxPATH
>JDBC driver property: ifxPATH.

>string

>false

ifxPDQPRIORITY
>JDBC driver property: ifxPDQPRIORITY.

>string

>false

ifxPLCONFIG
>JDBC driver property: ifxPLCONFIG.

>string

>false

ifxPLOAD_LO_PATH
>JDBC driver property: ifxPLOAD_LO_PATH.

>string

>false

ifxPROTOCOLTRACE
>JDBC driver property: ifxPROTOCOLTRACE.

>int

false

ifxPROTOCOLTRACEFILE
JDBC driver property: ifxPROTOCOLTRACEFILE.

string

false

ifxPROXY
JDBC driver property: ifxPROXY.

string

false

ifxPSORT_DBTEMP
JDBC driver property: ifxPSORT_DBTEMP.

string

false

ifxPSORT_NPROCS
JDBC driver property: ifxPSORT_NPROCS.

boolean

false

ifxSECURITY
JDBC driver property: ifxSECURITY.

string

false

ifxSQLH_FILE
JDBC driver property: ifxSQLH_FILE.

string

false

ifxSQLH_LOC
JDBC driver property: ifxSQLH_LOC.

string

false

ifxSQLH_TYPE
JDBC driver property: ifxSQLH_TYPE.

string

false

ifxSSLCONNECTION
JDBC driver property: ifxSSLCONNECTION.

string

false

ifxSTMT_CACHE
JDBC driver property: ifxSTMT_CACHE.

string

false

ifxTRACE
JDBC driver property: ifxTRACE.

int

false

ifxTRACEFILE
JDBC driver property: ifxTRACEFILE.

string

false

ifxTRUSTED_CONTEXT
JDBC driver property: ifxTRUSTED_CONTEXT.

string

false

ifxUSEV5SERVER
JDBC driver property: ifxUSEV5SERVER.

boolean

false

ifxUSE_DTENV
JDBC driver property: ifxUSE_DTENV.

boolean

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

roleName
JDBC driver property: roleName.

string

false

properties.informix.jcc

Data source properties for the IBM Data Server Driver for JDBC and SQLJ for Informix. PID is com.ibm.ws.jdbc.dataSource.properties.informix.jcc, and it is the child of complex type "dataSource".

databaseName
JDBC driver property: databaseName.

string

false

serverName
Server where the database is running.

string

localhost

false

portNumber
Port on which to obtain database connections.

int

1526

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

currentLockTimeout
JDBC driver property: currentLockTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

2s

false

DBANSIWARN
JDBC driver property: DBANSIWARN.

boolean

false

DBDATE
JDBC driver property: DBDATE.

string

false

DBPATH
JDBC driver property: DBPATH.

string

false

DBSPACETEMP
JDBC driver property: DBSPACETEMP.

string

false

DBTEMP
JDBC driver property: DBTEMP.

string

false

DBUPSPACE
JDBC driver property: DBUPSPACE.

string

false

DELIMIDENT
JDBC driver property: DELIMIDENT.

boolean

false

deferPrepares
JDBC driver property: deferPrepares.

boolean

false

driverType
JDBC driver property: driverType.

int

4

false

enableNamedParameterMarkers
JDBC driver property: enableNamedParameterMarkers. Values are: 1 (YES) or 2 (NO).

int

false

enableSeamlessFailover
JDBC driver property: enableSeamlessFailover. Values are: 1 (YES) or 2 (NO).

int

false

enableSysplexWLB
JDBC driver property: enableSysplexWLB.

boolean

false

fetchSize
JDBC driver property: fetchSize.

int

false

fullyMaterializeLobData
JDBC driver property: fullyMaterializeLobData.

boolean

false

IFX_DIRECTIVES
JDBC driver property: IFX_DIRECTIVES.

string

ON

OFF

false

IFX_EXTDIRECTIVES
> JDBC driver property: IFX_EXTDIRECTIVES.

> string

> **ON**

> **OFF**

> false

IFX_UPDDESC
> JDBC driver property: IFX_UPDDESC.

> string

> false

IFX_XASTDCOMPLIANCE_XAEND
> JDBC driver property: IFX_XASTDCOMPLIANCE_XAEND.

> string

> **0**

> **1**

> false

INFORMIXOPCACHE
> JDBC driver property: INFORMIXOPCACHE.

> string

> false

INFORMIXSTACKSIZE
> JDBC driver property: INFORMIXSTACKSIZE.

> string

> false

keepDynamic
> JDBC driver property: keepDynamic.

> int

> false

loginTimeout
> JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

> string

> false

NODEFDAC
> JDBC driver property: NODEFDAC.

> string

> **yes**

> **no**

false

OPTCOMPIND
    JDBC driver property: OPTCOMPIND.

    string

    0

    1

    2

    false

OPTOFC
    JDBC driver property: OPTOFC.

    string

    0

    1

    false

PDQPRIORITY
    JDBC driver property: PDQPRIORITY.

    string

    HIGH

    LOW

    OFF

    false

progressiveStreaming
    JDBC driver property: progressiveStreaming. Values are: 1 (YES) or 2 (NO).

    int

    1   YES

    2   NO

    false

PSORT_DBTEMP
    JDBC driver property: PSORT_DBTEMP.

    string

    false

PSORT_NPROCS
    JDBC driver property: PSORT_NPROCS.

    string

    false

queryDataSize
    JDBC driver property: queryDataSize.

    int

    false

resultSetHoldability

JDBC driver property: resultSetHoldability. Values are: 1 (HOLD_CURSORS_OVER_COMMIT) or 2 (CLOSE_CURSORS_AT_COMMIT).

int

1   HOLD_CURSORS_OVER_COMMIT

2   CLOSE_CURSORS_AT_COMMIT

false

resultSetHoldabilityForCatalogQueries

JDBC driver property: resultSetHoldabilityForCatalogQueries. Values are: 1 (HOLD_CURSORS_OVER_COMMIT) or 2 (CLOSE_CURSORS_AT_COMMIT).

int

1   HOLD_CURSORS_OVER_COMMIT

2   CLOSE_CURSORS_AT_COMMIT

false

retrieveMessagesFromServerOnGetMessage

JDBC driver property: retrieveMessagesFromServerOnGetMessage.

boolean

true

false

securityMechanism

JDBC driver property: securityMechanism. Values are: 3 (CLEAR_TEXT_PASSWORD_SECURITY), 4 (USER_ONLY_SECURITY), 7 (ENCRYPTED_PASSWORD_SECURITY), 9 (ENCRYPTED_USER_AND_PASSWORD_SECURITY).

short

3   CLEAR_TEXT_PASSWORD_SECURITY

4   USER_ONLY_SECURITY

7   ENCRYPTED_PASSWORD_SECURITY

9   ENCRYPTED_USER_AND_PASSWORD_SECURITY

false

STMT_CACHE

JDBC driver property: STMT_CACHE.

string

0

1

false

traceDirectory

JDBC driver property: traceDirectory.

string

false

traceFile

JDBC driver property: traceFile.

string

false

traceFileAppend
>JDBC driver property: traceFileAppend.

>boolean

>false

traceLevel
>Bitwise combination of the following constant values: TRACE_NONE=0,
>TRACE_CONNECTION_CALLS=1, TRACE_STATEMENT_CALLS=2,
>TRACE_RESULT_SET_CALLS=4, TRACE_DRIVER_CONFIGURATION=16, TRACE_CONNECTS=32,
>TRACE_DRDA_FLOWS=64, TRACE_RESULT_SET_META_DATA=128,
>TRACE_PARAMETER_META_DATA=256, TRACE_DIAGNOSTICS=512, TRACE_SQLJ=1024,
>TRACE_META_CALLS=8192, TRACE_DATASOURCE_CALLS=16384,
>TRACE_LARGE_OBJECT_CALLS=32768, TRACE_SYSTEM_MONITOR=131072,
>TRACE_TRACEPOINTS=262144, TRACE_ALL=-1.

>int

>false

useJDBC4ColumnNameAndLabelSemantics
>JDBC driver property: useJDBC4ColumnNameAndLabelSemantics. Values are: 1 (YES) or 2 (NO).

>int

>false

properties.microsoft.sqlserver

Data source properties for Microsoft SQL Server JDBC Driver. PID is
com.ibm.ws.jdbc.dataSource.properties.microsoft.sqlserver, and it is the child of complex type
"dataSource".

databaseName
>JDBC driver property: databaseName.

>string

>false

instanceName
>JDBC driver property: instanceName.

>string

>false

serverName
>Server where the database is running.

>string

>localhost

>false

portNumber
>Port on which to obtain database connections.

>int

>1433

false

user
Database user name.

string

false

password
Password for database user.

password (string)

false

applicationIntent
JDBC driver property: applicationIntent.

string

ReadOnly

ReadWrite

false

applicationName
JDBC driver property: applicationName.

string

false

authenticationScheme
JDBC driver property: authenticationScheme.

string

JavaKerberos

NativeAuthentication

false

encrypt
JDBC driver property: encrypt.

boolean

false

failoverPartner
JDBC driver property: failoverPartner.

string

false

hostNameInCertificate
JDBC driver property: hostNameInCertificate.

string

false

integratedSecurity
JDBC driver property: integratedSecurity.

boolean

false

lastUpdateCount
> JDBC driver property: lastUpdateCount.

> boolean

> false

lockTimeout
> JDBC driver property: lockTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

> string

> false

loginTimeout
> JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

> string

> false

multiSubnetFailover
> JDBC driver property: multiSubnetFailover.

> boolean

> false

packetSize
> JDBC driver property: packetSize.

> int

> false

responseBuffering
> JDBC driver property: responseBuffering.

> string

> **adaptive**

> **full**

> false

selectMethod
> JDBC driver property: selectMethod.

> string

> **cursor**

> **direct**

> false

sendStringParametersAsUnicode
> JDBC driver property: sendStringParametersAsUnicode.

> boolean

> false

false

sendTimeAsDatetime
    JDBC driver property: sendTimeAsDatetime.

    boolean

    false

trustServerCertificate
    JDBC driver property: trustServerCertificate.

    boolean

    false

trustStore
    JDBC driver property: trustStore.

    string

    false

trustStorePassword
    JDBC driver property: trustStorePassword.

    password (string)

    false

URL
    URL for connecting to the database. Example: jdbc:sqlserver://localhost:1433;databaseName=myDB.

    string

    false

workstationID
    JDBC driver property: workstationID.

    string

    false

xopenStates
    JDBC driver property: xopenStates.

    boolean

    false

properties.oracle

Data source properties for Oracle JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.oracle, and it is the child of complex type "dataSource".

driverType
    JDBC driver property: driverType.

    string

    thin

    thin

    oci

    false

databaseName
　　JDBC driver property: databaseName.

　　string

　　false

serverName
　　Server where the database is running.

　　string

　　localhost

　　false

portNumber
　　Port on which to obtain database connections.

　　int

　　1521

　　false

URL
　　URL for connecting to the database. Examples: jdbc:oracle:thin:@//localhost:1521/sample or
　　jdbc:oracle:oci:@//localhost:1521/sample.

　　string

　　false

user
　　Database user name.

　　string

　　false

password
　　Password for database user.

　　password (string)

　　false

connectionProperties
　　JDBC driver property: connectionProperties.

　　string

　　false

loginTimeout
　　JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be
　　hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include
　　multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

　　string

　　false

networkProtocol
　　JDBC driver property: networkProtocol.

　　string

　　false

ONSConfiguration
  JDBC driver property: ONSConfiguration.

  string

  false

serviceName
  JDBC driver property: serviceName.

  string

  false

TNSEntryName
  JDBC driver property: TNSEntryName.

  string

  false

properties.sybase

Data source properties for Sybase JDBC driver. PID is com.ibm.ws.jdbc.dataSource.properties.sybase, and it is the child of complex type "dataSource".

databaseName
  JDBC driver property: databaseName.

  string

  true

serverName
  Server where the database is running.

  string

  localhost

  false

portNumber
  Port on which to obtain database connections.

  int

  5000

  false

user
  Database user name.

  string

  false

password
  Password for database user.

  password (string)

  false

connectionProperties
  JDBC driver property: connectionProperties.

  string

SELECT_OPENS_CURSOR=true

false

loginTimeout
JDBC driver property: loginTimeout. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

false

networkProtocol
JDBC driver property: networkProtocol.

string

socket

SSL

false

resourceManagerName
JDBC driver property: resourceManagerName.

string

false

SERVER_INITIATED_TRANSACTIONS
JDBC driver property: SERVER_INITIATED_TRANSACTIONS.

string

true

false

false

version
JDBC driver property: version.

int

false

quickStartSecurity

Simple administrative security configuration. PID is com.ibm.ws.security.quickStartSecurity.

userName
Single user defined as part of the quick start security configuration. This user is granted the Administrator role.

string

true

userPassword
Password for the single user defined as part of the quick start security configuration. It is recommended that you encode this password. To do so, use the securityUtility tool with the encode option.

password (string)

true

safAuthorization

Controls the operation of the SAF Authorization Service. PID is com.ibm.ws.security.authorization.saf.

`roleMapper`
OSGi component name of the SAF Role Mapper service provider.

string

com.ibm.ws.security.authorization.saf.internal.SAFRoleMapperImpl

false

safCredentials

Controls the operation of the SAF Credentials Service. PID is com.ibm.ws.security.credentials.saf.

`unauthenticatedUser`
SAF user ID of the unauthenticated user.

string

WSGUEST

false

`profilePrefix`
Profile prefix used to specify the SAF APPL-ID when creating SAF credentials and authorizing users against SAF resource profiles.

string

BBGZDFLT

false

safRegistry

Configuration properties for a SAF user registry. PID is com.ibm.ws.security.registry.saf.config.

`realm`
The realm name represents the user registry.

string

false

safRoleMapper

Defines how to generate SAF EJBROLE resource profile names from application role names. PID is com.ibm.ws.security.authorization.saf.internal.SAFRoleMapperImpl.

`profilePattern`
Pattern to use for generating EJBROLE resource profile names from application role names.

string

%profilePrefix%.%resource%.%role%

false

`toUpperCase`
Convert the EJBROLE resource profile name to upper-case.

boolean

false

false

securewayLdapFilterProperties

Specifies the list of default SecureWay LDAP filters. PID is
com.ibm.ws.security.registry.ldap.internal.filters.secureway.

userFilter
> An LDAP filter clause for searching the user registry for users.
>
> string
>
> (&(uid=%v)(objectclass=ePerson))
>
> true

groupFilter
> An LDAP filter clause for search the user registry for groups.
>
> string
>
> (&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))
>
> true

userIdMap
> An LDAP filter that maps the name of a user to an LDAP entry.
>
> string
>
> *:uid
>
> true

groupIdMap
> groupIdFilter.desc
>
> string
>
> *:cn
>
> true

groupMemberIdMap
> An LDAP filter that identifies user to group memberships.
>
> string
>
> groupOfNames:member;groupOfUniqueNames:uniqueMember
>
> true

ssl

An SSL configuration repertoire with an ID, a defined keystore, and an optional truststore. PID is
com.ibm.ws.ssl.repertoire.

keyStoreRef
> A keystore containing key entries for the SSL configuration repertoire. This attribute is required.
>
> string
>
> true

trustStoreRef
> A keystore containing trusted certificate entries used by the SSL configuration repertoire for signing
> verification. This attribute is optional. If unspecified, the same keystore is used for both key and trusted
> certificate entries.
>
> string
>
> ${keyStoreRef}

false

sslDefault

The default configuration repertoire for SSL services. PID is com.ibm.ws.ssl.default.

sslRef
  The default SSL configuration repertoire. The default value is defaultSSLSettings.

  string

  defaultSSLConfig

  false

sslOptions

The SSL protocol configuration for a transport. PID is com.ibm.ws.sslchannel.options.

sessionTimeout
  Amount of time to wait for a read or write request to complete on a socket. This value is overridden by protocol-specific timeouts. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

  string

  1d

  false

sslRef
  default.sslRef.desc

  string

  false

tcpOptions

Defines TCP protocol settings. PID is com.ibm.ws.tcpchannel.options.

inactivityTimeout
  Amount of time to wait for a read or write request to complete on a socket. This value is overridden by protocol-specific timeouts. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

  string

  60s

  false

soReuseAddr
  Enables immediate rebind to a port with no active listener.

  boolean

  false

  false

transaction

Configuration properties for the Transaction Manager service. PID is com.ibm.ws.transaction.

recoverOnStartup

    Specifies whether the server should begin transaction recovery at server startup.

    boolean

    false

    false

waitForRecovery

    Specifies whether the server should wait for transaction recovery to complete before accepting new transactional work.

    boolean

    false

    false

acceptHeuristicHazard

    Specifies whether all applications on this server accept the possibility of a heuristic hazard occurring in a two-phase transaction that contains a one-phase resource.

    boolean

    true

    false

clientInactivityTimeout

    Maximum duration between transactional requests from a remote client. Any period of client inactivity that exceeds this timeout results in the transaction being rolled back in this application server. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

    string

    60s

    false

heuristicRetryInterval

    Amount of time that the application server waits before retrying a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

    string

    60s

    false

heuristicRetryWait

    The number of times that the application server retries a completion signal, such as commit or rollback. Retries occur after a transient exception from a resource manager or remote partner.

    int

    5

    false

propogatedOrBMTTranLifetimeTimeout

    Upper limit of the transaction timeout for transactions that run in this server. This value should be greater than or equal to the value specified for the total transaction timeout. Specify a positive integer

followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

0

false

totalTranLifetimeTimeout
Default maximum time allowed for transactions started on this server to complete. Any such transactions that do not complete before this timeout occurs are rolled back. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.

string

12000s

false

transactionLogDirectory
A directory for this server where the transaction service stores log files for recovery.

string

${server.config.dir}/tranlog/

false

transactionLogSize
Specifies the size of transaction log files in Kilobytes.

int

1024

false

enableLoggingForHeuristicReporting
Specifies whether the application server logs about-to-commit-one-phase-resource events from transactions that involve both a one-phase commit resource and two-phase commit resources.

boolean

false

false

timeoutGracePeriodEnabled
Specifies whether there is a delay between a transaction timeout and the abnormal ending of the servant region that was running the transaction.

boolean

false

false

lpsHeuristicCompletion
Specifies the direction that is used to complete a transaction that has a heuristic outcome; either the application server commits or rolls back the transaction, or depends on manual completion by the administrator. Allowed values are: COMMIT, ROLLBACK and MANUAL

string

ROLLBACK

ROLLBACK

COMMIT

MANUAL

false

`defaultMaxShutdownDelay`
>Default maximum shutdown delay. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds.
>
>string
>
>2s
>
>false

trustAssociation

Controls the operation of the trust association interceptor (TAI) service. PID is com.ibm.ws.security.authentication.tai.

`invokeForUnprotectedURI`
>Controls whether the TAI is invoked for an unprotected URI.
>
>boolean
>
>false
>
>true

`failOverToAppAuthType`
>Allow an interceptor to fall back to the application authentication mechanism.
>
>boolean
>
>false
>
>true

`interceptors`
>Defines a trust association interceptor.
>
>>`enabled`
>>>Enables or disables the interceptor.
>>>
>>>boolean
>>>
>>>true
>>>
>>>true
>>
>>`className`
>>>Fully-qualified package name of the interceptor class.
>>>
>>>string
>>>
>>>true
>>
>>`invokeBeforeSSO`
>>>Invoke an interceptor before single sign-on (SSO).
>>>
>>>boolean
>>>
>>>true
>>>
>>>true

invokeAfterSSO
  Invoke an interceptor after single sign-on (SSO).

  boolean

  false

  true

libraryRef
  A reference to the ID of the shared library configuration.

  Configuration ID of type library (string).

  false

library
  A reference to the ID of the shared library configuration.

  Element of type library.

  false

properties
  false

  true

virtualHost

Virtual host configuration. PID is com.ibm.ws.http.virtualhost.

enabled
  Enable this virtual host.

  boolean

  true

  false

webAppSecurity

Configures web container application security. PID is
com.ibm.ws.webcontainer.security.internal.WebAppSecurityCollaboratorImpl.

allowFailOverToBasicAuth
  Specifies whether to fail over to basic authentication when certificate authentication fails. The
  equivalent custom property in the full application server profile is
  com.ibm.wsspi.security.web.failOverToBasicAuth.

  boolean

  false

  false

allowLogoutPageRedirectToAnyHost
  Warning, security risk: Setting this property to true may open your systems to potential URL redirect
  attacks. If set to true, any host can be specified for the logout page redirect. If set to false, and the
  logout page points to a different host, or one not listed in the logout page redirect domain list, then a
  generic logout page is displayed. The equivalent custom property in the full application server profile is
  com.ibm.websphere.security.allowAnyLogoutExitPageHost.

  boolean

  false

false

displayAuthenticationRealm

Warning, security risk: if this property is set to true, and the user registry's realm name contains sensitive information, it is displayed to the user. For example, if an LDAP configuration is used, the LDAP server hostname and port are displayed. This configuration controls what the HTTP basic authentication login window displays when the realm name is not defined in the application web.xml. If the realm name is defined in the application web.xml file, this property is ignored. If set to true, the realm name displayed will be the user registry realm name for the LTPA authentication mechanism or the Kerberos realm name for the Kerberos authentication mechanism. If set to false, the realm name displayed will be "Default Realm". The equivalent custom property in the full application server profile is com.ibm.websphere.security.displayRealm.

boolean

false

false

httpOnlyCookies

Specifies whether the HTTP only (HttpOnly) cookies option is enabled.

boolean

true

false

logoutOnHttpSessionExpire

Specifies whether users will be logged out after the HTTP session timer expires. If set to false, the user credential will stay active until the Single Sign-On token timeout occurs. The equivalent custom property in the full application server profile is com.ibm.ws.security.web.logoutOnHTTPSessionExpire.

boolean

false

false

logoutPageRedirectDomainNames

A pipe (l) separated list of domain names that are allowed for the logout page redirect (localhost is implied). The equivalent custom property in the full application server profile is com.ibm.websphere.security.logoutExitPageDomainList.

string

false

singleSignonEnabled

Specifies whether single sign-on is enabled.

boolean

true

false

ssoDomainNames

A pipe (l) separated list of domain names that SSO Cookies should be presented. The equivalent custom property in the full application server profile is com.ibm.ws.security.config.SingleSignonConfig

string

false

ssoUseDomainFromURL

Specifies whether to use the domain name from the request URL for the cookie domain.

boolean

false

false

ssoRequiresSSL
Specifies whether a SSO cookie is sent over SSL. The equivalent custom property in the full application server profile is com.ibm.websphere.security.customSSOCookieName

boolean

false

false

preserveFullyQualifiedReferrerUrl
Warning, security risk: Setting this to true may open your systems to potential URL redirect attacks. This property specifies whether the fully qualified referrer URL for form login redirects is preserved. If false, the host for the referrer URL is removed and the redirect is to localhost. The equivalent custom property in the full application server profile is com.ibm.websphere.security.util.fullyQualifiedURL

boolean

false

false

postParamSaveMethod
Specifies where POST parameters are stored upon redirect. Valid values are cookie (POST parameters are stored in a cookie), session (POST parameters are stored in the HTTP Session) and none (POST parameters are not preserved). The equivalent custom property in the full application server profile is com.ibm.websphere.security.util.postParamSaveMethod.

string

Cookie

Cookie

Session

None

false

postParamCookieSize
Size of the POST parameter cookie. If the size of the cookie is larger than the browser limit, unexpected behavior may occur. The value of this property must be a positive integer and represents the maximum size of the cookie in bytes. The equivalent custom property in the full application server profile is com.ibm.websphere.security.util.postParamMaxCookieSize.

int

16384

false

ssoCookieName
Customizes the SSO cookie name. A custom cookie name allows you to logically separate authentication between SSO domains and to enable customized authentication to a particular environment. Before setting this value, consider that setting a custom cookie name can cause an authentication failure. For example, a connection to a server that has a custom cookie property set sends this custom cookie to the browser. A subsequent connection to a server that uses either the default cookie name or a different cookie name, is not able to authenticate the request via a validation of the in-bound cookie. The equivalent custom property in the full application server profile is com.ibm.websphere.security.customSSOCookieName.

string

LtpaToken2

false

webAlwaysLogin
Specifies whether the login() method will throw an exception when an identity has already been authenticated.

boolean

false

false

useAuthenticationDataForUnprotectedResource
Specifies whether authentication data can be used when accessing an unprotected resource. The unprotected resource can access validated authenticated data that it previously could not access. This option enables the unprotected resource to call the getRemoteUser, isUserInRole, and getUserPrincipal methods to retrieve an authenticated identity. The equivalent custom property in the full application server profile is com.ibm.wsspi.security.web.webAuthReq=persisting.

boolean

true

false

webContainer

Configuration for the web container. PID is com.ibm.ws.webcontainer.

listeners
A comma separated list of listener classes.

string

false

decodeUrlAsUtf8
Decode URL using the an encoding setting of UTF-8.

boolean

true

false

fileServingEnabled
Enable file serving if this setting was not explicitly specified for the application.

boolean

true

false

disallowAllFileServing
Disables all file serving by applications. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.disallowAllFileServing.

boolean

false

false

directoryBrowsingEnabled
> Enable directory browsing of an application.

> boolean

> false

> false

serveServletsByClassnameEnabled
> Enable servlets to be accessed in a web application using a class name if not explicitly specified.

> boolean

> false

> false

disallowServeServletsByClassName
> Disallows the use of serveServletsByClassnameEnabled on the application server level. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.disallowserveservletsbyclassname.

> boolean

> false

> false

doNotServeByClassName
> A semi-colon delimited list of classes to be completely disallowed from being served by classname. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.donotservebyclassname.

> string

> false

trustHostHeaderPort
> Set this property to true and the com.ibm.ws.webcontainer.extractHostHeaderPort custom property to true to return the port number from the request host header first.

> boolean

> false

> false

trusted
> Enables the application server to use inbound private headers from the web server plug-in.

> boolean

> true

> false

extractHostHeaderPort
> The web container will return a port number from the host header, if any, or the URL port on which the client connection was accepted. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.extracthostheaderport.

> boolean

> false

> false

httpsIndicatorHeader
> For SSL offloading, set to the name of the HTTP header variable inserted by the SSL accelerator/proxy/load balancer.
>
> string
>
> false

exposeWebInfOnDispatch
> If true, a servlet can access files in the WEB-INF directory. If false (default), a servlet cannot access files the WEB-INF directory.
>
> boolean
>
> false
>
> false

decodeUrlPlusSign
> Decode the plus sign when it is part of the URL. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.decodeurlplussign.
>
> boolean
>
> false
>
> false

channelWriteType
> When set to 'sync', responses will be written synchronously; otherwise, responses will be written asychronously. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.channelwritetype.
>
> string
>
> async
>
> false

suppressHtmlRecursiveErrorOutput
> Suppresses the HTML output of a recursive error that cannot be handled by an application's configured error page. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.suppressHtmlRecursiveErrorOutput.
>
> boolean
>
> false
>
> false

fileWrapperEvents
> Web container will generate SMF and PMI data when serving the static files. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.fileWrapperEvents.
>
> boolean
>
> false
>
> false

webGroupVhostNotFound
> Replaces message SRVE0017W or SRVE0255E with a user-defined string. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.webgroupvhostnotfound.
>
> string
>
> false

defaultTraceRequestBehavior
Restore HTTP TRACE processing. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.DefaultTraceRequestBehavior.

boolean

false

false

defaultHeadRequestBehavior
Restore the behavior where the HEAD request is not subject to the security constraint defined for the GET method. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.DefaultHeadRequestBehavior.

boolean

false

false

tolerateSymbolicLinks
Enables the web container to support the use of symbolic links. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.TolerateSymbolicLinks.

boolean

false

false

symbolicLinksCacheSize
Initial size of the symbolic link cache. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.SymbolicLinksCacheSize.

int

1000

false

enableErrorExceptionTypeFirst
Web container is updated to search and use the exception-type before the error-code. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.enableErrorExceptionTypeFirst.

boolean

false

false

enableMultiReadOfPostData
Retain post data for multiple read accesses. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.enablemultireadofpostdata.

boolean

false

false

copyAttributesKeySet
Web container will return an enumeration of a copy of the list attributes to the servlet to avoid a concurrent access error by the servlet. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.copyattributeskeyset.

boolean

false

false

dispatcherRethrowsEr
Web container will re-throw errors allowing interested resources to process them. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.dispatcherRethrowser.

boolean

true

false

ignoreSessiononStaticFileRequest
Improves performance by preventing the web container from accessing a session for static file requests involving filters. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.IgnoreSessiononStaticFileRequest.

boolean

false

false

invokeFilterInitAtStartup
Web container will call the filter's init() method at application startup. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.invokeFilterInitAtStartup.

boolean

true

false

enableJspMappingOverride
Allow the JSP mapping to be overridden so that the application can serve the JSP contents itself. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.enablejspmappingoverride.

boolean

false

false

enableDefaultIsElIgnoredInTag
Correct the default behavior of the EL expression's evaluation in the tag files. The equivalent custom property in the full application server profile is com.ibm.ws.jsp.enabledefaultiselignoredintag.

boolean

false

false

parseUtf8PostData
Web container will detect non URL encoded UTF-8 post data and include it in the parameter values. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.parseutf8postdata.

boolean

false

false

logServletContainerInitializerClassLoadingErrors
Log servlet container class loading errors as warnings rather than logging them only when debug is

enabled. The equivalent custom property in the full application server profile is
com.ibm.ws.webcontainer.logservletcontainerinitializerclassloadingerrors.

boolean

false

false

allowIncludeSendError
Allow RequestDispatch to send errors on Include methods. The equivalent custom property in the full
application server profile is com.ibm.ws.webcontainer.allowincludesenderror.

boolean

false

false

skipMetaInfResourcesProcessing
Do not search the meta-inf directory for application resources. The equivalent custom property in the
full application server profile is com.ibm.ws.webcontainer.skipmetainfresourcesprocessing.

boolean

false

false

metaInfResourcesCacheSize
Initial size (number of entries) of the meta-inf resource cache. The equivalent custom property in the
full application server profile is com.ibm.ws.webcontainer.metainfresourcescachesize.name.

int

20

false

xPoweredBy
Alternative string for the X-Powered-By header setting. The equivalent custom property in the full
application server profile is com.ibm.ws.webcontainer.xpoweredby.

string

false

disableXPoweredBy
Disable setting of the X-Powered-By header. The equivalent custom property in the full application
server profile is com.ibm.ws.webcontainer.disablexpoweredby.

boolean

false

false

deferServletLoad
Defer servlet loading and initialization until the first request.

boolean

true

false

asyncMaxSizeTaskPool
Maximum size of tasks in the Async task pool before automatically purging canceled tasks. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.asyncmaxsizetaskpool.

int

5000

false

asyncPurgeInterval
Time interval to wait between each required purge of the cancelled task pool. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.asyncpurgeinterval.

int

30000

false

asyncTimeoutDefault
Async servlet timeout value used when a timeout value has not been explcitly specified. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.asynctimeoutdefault.

int

30000

false

asyncTimerThreads
Maximum number of threads to use for async servlet timeout processing. The equivalent custom property in the full application server profile is com.ibm.ws.webcontainer.asynctimerthreads.

int

2

false

wlmClassification

zos.wlm.classification.config.description. PID is com.ibm.ws.zos.wlm.classification.

classificationName
optional classification name

string

null

false

zosWorkloadManager

WLM Native Enclave Manager Configuration. PID is com.ibm.ws.zos.wlm.WLMConfigManager.

collectionName
A collection name to be used for classifying work requests. The name is supplied to native services IWMCLSFY or __server_classify. Only the first 18 characters will be used.

string

${wlp.server.name}

false

Editing the Liberty profile using developer tools

You can modify how the workbench interacts with the Liberty profile by using the server editor.

Procedure

1. In the Servers view, right-click the server and select **Open**.
2. The server editor opens.

What to do next

You can modify the publishing, timeout and other settings regarding the interaction between the workbench and the server.

Editing the Liberty profile configuration using developer tools

You can modify the behavior of the Liberty profile by editing the configuration. For example, you can configure which HTTP ports to use, what features are enabled, and logging and tracing settings.

Before you begin

For a description of the underlying process of configuring a server, and detailed information about specific aspects of server configuration, see "Configuring the Liberty profile runtime environment" on page 130.

Procedure

1. To open the Server Configuration editor, select any of the following options:
   - In the **Servers** view, right-click the server configuration and select **Open**.
   - In the **Project Explorer** view, expand your server project and the server folder. Right-click the server.xml file and select **Open**.
2. Under the **Configuration Structure** section the elements in the configuration are displayed.
3. Under the **Feature Manager Details** section the details for the currently selected element are displayed.
4. To modify the details of an element, select the element under the **Configuration Structure** section and use the controls under the **Feature Manager Details** section to make modifications.
5. To add new elements, select **Server Configuration** under the **Configuration Structure** section then click **Add**.
6. To add child elements, select the parent element under the **Configuration Structure** section and then click **Add**.
7. You can remove or move elements by selecting the element and using the **Remove**, **Up**, and **Down** buttons.

Defining a utility project as a shared library

You can define a utility project as a shared library and associate defined shared libraries with an application or web project.

Before you begin

To use the shared library function in the workbench, you must create a utility project and define it as a shared library. The utility project is the only project type that can be used as a shared library.

About this task

A shared library is an external Java archive (JAR) file that is used by one or more applications. Using shared libraries enables multiple application published on a server to use a single library, rather than use

multiple copies of the same library. After you associate shared libraries with an application or project, the application or module class loader loads classes in the shared libraries and makes those classes available to the application or module.

Procedure

To define a utility project as a shared library:

1. Create a utility project:
   a. In the toolbar, select **File** > **New** > **Project**.
   b. Expand **Java EE** and select **Utility Project**. Click **Next**.
   c. In the **Project name** field, specify a name for the utility project.
   d. Under the **Target runtime** section, verify the WebSphere® Application Server Liberty profile is selected.
   e. Under the **Ear membership** section, clear the **Add project to an EAR** check box.
   f. Click **Finish**.
2. Define the artifacts in the newly created utility project. For example, you can add Java classes to the utility project.
3. Define the utility project as a shared library:
   a. In the **Project Explorer** view, right-click the utility project and select **Properties** > **Liberty Profile Shared Library**.
   b. In the **Shared library ID** field, type a string as an identifier for the shared library.
   c. In the **Archive directory** field, type or browse to a directory where you want to place the compressed copy of your utility project as a JAR file. The file name convention of the JAR file is *utilityProjectName*.jar, where *utilityProjectName* is the name of the utility project.
   d. In the **Liberty Profile Shared Library** page, click **Apply** to confirm your changes. Click **OK** to close the Properties window.
4. Add the utility project to the server. For more details see Adding and running an application on the Liberty profile using developer tools topic.

Results

Here is an example entry added to the server configuration (`server.xml`) file:

`<library id="libid"><fileset dir="C:\temp" includes="Util.jar"/></library>`

In addition, the JAR file is added in the specified archive directory. In the previous example, the `Util.jar` file is added in the `C:\temp` directory.

Setting a web project to use shared libraries
If you have a utility project defined as a shared library, you can associate defined shared libraries with a web project.

Before you begin
- Define a utility project as a shared library.

About this task

A shared library is an external Java archive (JAR) file that is used by one or more applications. Using shared libraries enables multiple application published on a server to use a single library, rather than use multiple copies of the same library. After you associate shared libraries with an application or project, the application or module class loader loads classes in the shared libraries and make those classes available to the application or module.

Procedure

1. To set a web project to use shared libraries:

   a. In the **Project Explorer** view, right-click your web project that you want to associate shared libraries.

   b. Select **Properties** > **Liberty Profile Shared Libraries**.

   c. In the **IDs** field, specify one or more shared library identifiers that you want the project to reference. To specify multiple identifiers, use a comma-separated list. For example: ID1, ID2, ID3

      Tip: The shared library identifier is the value specified in the **Shared Libary ID** field from the Defining a utility project as a shared library task.

2. You might want to add its associating utility projects to the class path for compilation-purpose:

   a. In the **Project Explorer** view, right-click your project that you are associating shared libraries.

   b. Select **Properties** > **Java Build Path**.

   c. Select the **Projects** tab.

   d. Click **Add**.

   e. Select the utility projects that the project references.

3. Develop the artifacts in the web project. For example, you can add a servlet in a web project that references classes in the shared libraries.

4. Add the web project to the server. For more details see Adding and running an application on the Liberty profile using developer tools topic.

Results

Here is an example entry added to the server configuration (`server.xml`) file:

```
<application type="war" id="web" name="web" location="web.war">
 <classloader commonLibraryRef="libid"/>
</application>
```

Exploring the runtime environment

You can use the **WebSphere Runtime Explorer** view to browse the available servers as well as any shared configurations and applications for a runtime environment. This view shows all of the available servers for the runtime environment as opposed to the **Servers** view which shows only those servers that are configured in the workspace. In both the WebSphere Runtime Explorer and Servers view, you can expand each server to show the configuration for that server.

Before you begin

The application server that the WebSphere Runtime Explorer view supports is the WebSphere Application Server Liberty profile.

About this task

You can use the WebSphere Runtime Explorer view to complete the following tasks:
- View the servers defined in a runtime environment
- View the shared applications defined in a runtime environment
- View the shared configurations files defined in a runtime environment
- Create a new runtime environment
- Create a new server in a runtime environment

Procedure

To open the WebSphere Runtime Explorer view:
1. In the toolbar of the workbench, select **Window** > **Show View** > **Other**.
2. In the **Show View** window, expand **Server** and select **WebSphere Runtime Explorer**. Click **OK**.

Displaying the server configuration in a merged view

You can use the **Merged Configuration** view to see a flattened view of the server configuration and any recursively included configuration files.

About this task

In the Server Configuration editor, under the **Configuration Structure** section, you can use the **Include** element to import files that contain additional configuration settings. The **Include** element can embed multiple layers of configuration files within the server.xml file, which can make the configuration difficult to read without tools. The **Merged Configuration** view provides a flattened view of the server configuration and any recursively included configuration files. This is a read-only view and cannot be edited.

Procedure

To display the server configuration in a merged view:

In the Servers view, right-click the server configuration and select **Open Merged View**.

Example

Here is an example of the source code for the server.xml file. Look at the **include** tag which imports the common.xml file:

```
<server>
 <featureManager>
  <feature>jsp-2.2</feature>
  <feature>servlet-3.0</feature>
 </featureManager>
 <application id="Web2.5" location="Web2.5.war" name="Web2.5" type="war"/>
 <include location="common.xml"/>
</server>
```

Here is an example of the source code for the common.xml file:

```
<server>
 <application id="Setup" location="Setup.war" name="Setup" type="war"/>
</server>
```

Look under the **Configuration Structure** sections to see the difference between the Server Configuration editor and Merged Configuration view. The Merged Configuration view replaces the **Include: common.xml** element from the Server Configuration editor with **Application: Setup** element.

Viewing the schema documentation for the server configuration

You can view the schema documentation for the server configuration (server.xml file) within the workbench. The documentation provides information about the configuration elements that are available, the default settings, and details for each of the elements.

Procedure

In the **Servers** view, right-click on the server configuration and select **Open Schema Reference**.

Configuring the Liberty profile runtime environment

A server configuration consists of a `server.xml` file, a `bootstrap.properties` file, and any optional files that are included by the two main configuration files. There is no administrative console for the Liberty profile. Administrators and developers use the Liberty profile developer tools, or a text editor, to edit the configuration files.

About this task

The `server.xml` file is the primary configuration file for the server. You can edit this file, and the files it includes, in a text editor. Alternatively, you can use the editor that is part of the Liberty profile developer tools. You can also change the location of the `server.xml` file. However, for most configurations you do not need to do this.

The `bootstrap.properties` file is used to specify properties that need to be available before the main configuration is processed. If you update the `bootstrap.properties` file, you must restart the server with the `--clean` option for the changes to take effect.

Procedure

- Specify server configuration properties.

  All the elements that can be configured in the `server.xml` file, and the files it includes, are described in "Liberty profile: Configuration elements in the server.xml file" on page 3. However, the only required element is a server definition:

  ```
  <server/>
  ```

  Beyond this server definition, you only specify overrides and additions to the default configuration values. For example, to change the transaction timeout value, you specify:

  ```
  <transactions timeout="30" />
  ```

  Some attributes can have multiple values. For example, you use a list of values to define the features that are to be provided by the server:

  ```
  <server>
      <featureManager>
          <feature>servlet-3.0</feature>
          <feature>localConnector-1.0</feature>
      </featureManager>
  </server>
  ```

  See also "Adding and removing server features" on page 137.

  Where multiple instances of a resource type can be configured, for example applications or data sources, you need only provide the attributes that are unique for the resource. You can let the other attributes use the default values, or override them as needed. Therefore the contents of the `server.xml` file can be brief. For example, here is a complete server configuration to run a web application:

  ```
  <server>
      <featureManager>
          <feature>servlet_3.0</feature>
      </featureManager>
      <application name="snoop" location="/mywebapps/snoop" id="snoop" type="war"/>
  </server>
  ```

  For detailed information about specific aspects of server configuration, see the following topics:
  - "Using includes in the configuration" on page 138
  - "Using variables in the configuration" on page 140
  - "Controlling dynamic updates" on page 141
- Specify when servlets are loaded and initialized.
- Configure the Liberty profile with a web server plug-in.

Customizing the Liberty profile environment

You can customize the Liberty profile environment using certain specific variables to support the placement of product binaries and shared resources in read-only file systems.

About this task

The following Liberty profile specific variables can be used to customize the Liberty profile environment:

- `${wlp.install.dir}`

    This configuration variable has an inferred location. The installation directory is always set to the parent of the directory containing the launch script or the parent of the `/lib` directory containing the target jar files.

- `WLP_USER_DIR`

    This environment variable can be used to specify an alternate location for `${wlp.user.dir}`. This variable can only be an absolute path. If this is specified, the runtime environment looks for shared resources and server definition in the specified directory. The `${server.config.dir}` is equivalent to `${wlp.user.dir}/servers/serverName`.

- `WLP_OUTPUT_DIR`

    This environment variable can be used to specify an alternate location for server generated output such as logs, the `workarea` directory and generated files. This variable can only be an absolute path. If this environment variable is specified, `${server.output.dir}` is set to the equivalent of `WLP_OUTPUT_DIR/serverName`. If not, the `${server.output.dir}` is the same as `${server.config.dir}` .

You can specify `WLP_OUTPUT_DIR` and `WLP_USER_DIR` environment variables in `server.env` files. You can also specify JVM options in `jvm.options` files.

Procedure

- Specify environment variables by using `server.env` files.

    You can use `server.env` files at the installation and server levels to specify environment variables such as `JAVA_HOME`, `WLP_USER_DIR` and `WLP_OUTPUT_DIR`. For example:

    `JAVA_HOME=/opt/ibm/java-i386-60/jre`

    Note:

    - The `server.env` files support only `key=value` pairs.
    - Shell and variable expansion are not supported.
    - `WLP_USER_DIR` can only be specified in the `${wlp.install.dir}/etc/server.env` file because the purpose of this variable is to specify where the remaining configuration is located. Once the remaining configuration is found and merged, no further configuration in a different location is expected, or supported.

    The server management script searches for `server.env` files in two locations: `${wlp.install.dir}/etc/server.env` and `${server.config.dir}/server.env` files. If both files are present, the contents of the two files are merged with values in the server-level file taking precedence over values in the runtime-level file.

    You can also specify these environment variables in the shell environment, but the `server.env` files take precedence over those variables.

- Customize JVM options by using `jvm.options` files.

    You can use `jvm.options` files at the runtime and server levels to specify additional server startup options, for example, **-X** arguments. The options are applied when the start, run, and debug actions are invoked through the server management script. Be sure to specify only one option per line. For example:

```
# Set the maximum heap size to 1024m.
-Xmx1024m
# Set a system property.
-Dcom.ibm.example.system.property=ExampleValue
# Enable verbose output for class loading.
-verbose:class
```

The server management script searches for `jvm.options` in two locations: `${wlp.install.dir}/etc/jvm.options` and `${server.config.dir}/jvm.options` files. If both files are present, the options in the `${server.config.dir}/jvm.options` file are used.

Administering the Liberty profile from the command prompt

You can use the `server` and `ws-launch.jar` commands to create a server, to start or stop a server, to check if it is running, and to help debug it.

About this task

The `wlp/bin` directory contains a script called `server` to help with controlling the server process. The syntax of this script is as follows:

```
server <task> [server] [options]
```

This script supports the following operations:

create A command that creates a new server.

run A command that launches the server in the foreground.

debug A command that runs the named server in the console foreground after a debugger connects to the debug port. The default port is 7777.

dump A command that creates a snapshot of a server and saves the result into an archive file for further tuning and diagnosis.

package
A command that packages a server.

start A command that launches the server as a background process.

stop A command that stops a running server.

status A command that checks to see if a specified server is running.

version
A command that displays the version information of current server and Java runtime environment.

help A command that gets command-line script help, including details of additional options.

Note: If a server is not specified on the command line, the task is performed against the default server instance, `defaultServer`, if it exists.

You can also carry out similar tasks using the `ws-launch.jar` script that is in the `wlp/lib` directory.

Example

To run the `server` script on Windows systems:

```
bin\server.bat create server_name
bin\server.bat package server_name
bin\server.bat run server_name
bin\server.bat help server_name
```

To run the `server` script on other systems:

```
bin/server create server_name
bin/server package server_name
bin/server run server_name
bin/server help server_name
```

To run the `ws-launch.jar` script:

```
java -javaagent:lib/bootstrap-agent.jar -jar lib/ws-launch.jar server_name --create
java -javaagent:lib/bootstrap-agent.jar -jar lib/ws-launch.jar server_name
java -javaagent:lib/bootstrap-agent.jar -jar lib/ws-launch.jar --help
```

The `--help` option provides information about additional command line parameters for the `ws-launch.jar` script, such as `--stop, --version, --clean, --init-log, --include`.

Liberty profile: server command

The command supports starting, stopping, creating, packaging and dumping a Liberty profile server.

Syntax

The command syntax is as follows:

```
server task serverName [options]
```

where the value of **task** represents the operation that you can perform on a Liberty profile server. See available administration operations for the Liberty profile from the command prompt.

Options

The following options are available for the **server** command:

--archive="path to the target archive file"
> Specifies a target file for the *package* operation. This path could be either a relative path, which is relative to the installation root directory of the Liberty profile, or an absolute path. The default archive target is a compressed file with the server name, which will be stored in the installation root directory. Remember to use quotes if the value contains spaces. You can use this option for both *package* and *dump* operations.

--clean
> Cleans all cached information related to this server instance.

--include=all/usr
> Specifies whether packaging all the files in the Liberty profile installation directory or only the files in `/usr` directory. You can only use `all` or `user` value for the *package* operation. If the `${WLP_USER_DIR}` and `${WLP_OUTPUT_DIR}` are defined in the `server.env` file, the files under them are packaged.

> **Note:** Files referenced in the server configuration file are not processed, therefore they might not be packaged.

Usage

The following examples demonstrate correct syntax:

```
server start myserver --clean
server package myserver --archive="c:\mybackup\" --include=all
server dump myserver --archive="c:\mybackup\" --include=heap
```

Using Ant to automate task for the Liberty profile

Apache Ant is a Java library tool for automating the build process. You can use Ant tasks provided by the Liberty profile to manage the server and applications.

Before you begin

The Ant plug-in for the Liberty profile is located under the `dev/tools/ant` directory of the server image. This plugin contains a set of Ant tasks. If you want to use these tasks in your build script, you must make sure the plug-in is available on the Ant classpath. One way of doing this is to copy the plug-in file `wlp-anttasks.jar` to the `/lib` directory of the Ant installation, and declare the `antlib` namespace in the `build.xml` file. For example:

```
<project .... xmlns:wlp="antlib:com.ibm.websphere.wlp.ant">
  ...
</project>
```

The namespace can be any string, provided you avoid name conflicts. After that, you must use the namespace as a prefix of the Ant tasks for the Liberty profile. For example, you must use `wlp:server` when calling the `server` task.

About this task

You can create build scripts that use these Ant tasks to package, install, and test your application on the Liberty profile.

Procedure
- Manage the Liberty profile server.
- Install applications on a Liberty profile server.
- Remove application from a Liberty profile server.

Liberty profile: Ant task - server:

The `server` task can be used to manage the status of a Liberty profile server.

Description

The `server` task supports the following options to manage the status of a Liberty profile server:
- **create**, which creates a named server instance.
- **start**, which starts the named server instance. If the server instance does not exist, this option creates one by default.
- **stop**, which stops the named server.
- **status**, which checks the server status.
- **package**, which packages the named server and its deployed applications.

Attributes

The table below describes attributes of the `server` task.

Table 1. The attributes of the `server` task.

The first column contains a list of attributes, the second column contains a description of each attribute, and the third column represents whether this attribute is required.

Attribute	Description	Required
`installDir`	Location of the Liberty profile server.	Yes
`operation`	Server operations available as options: **create**, **start**, **stop**, **status**, and **package**.	Yes

Table 1. The attributes of the `server` *task (continued).*

The first column contains a list of attributes, the second column contains a description of each attribute, and the third column represents whether this attribute is required.

Attribute	Description	Required
`serverName`	Name of the Liberty profile server instance. The default value is `defaultServer`.	No
`userDir`	Value of the `${wlp_user_dir}` variable. The default value is `${installDir}/user/servers/${serverName}`.	No
`outputDir`	Value of the `${wlp_output_dir}` variable. The default value is `${installDir}/user/servers/${serverName}`	No
`clean`	Whether to operate the server using the `clean` option.	No
`timeout`	Waiting time before the server starts or stops. The default value is 30 seconds.	No
`archive`	Location of compressed file when packaging a server. Only available to the `package` option. The value must be a file name and only works for `package` option.	No
`ref`	Reference to an existing server task definition to reuse its server configuration. The value can be null when other required attributes are set.	No

Example

This example shows how to use the `server` task in your `build.xml` file:

```
<wlp:server id="wlp.ant.test" installDir="${wlp_install_dir}" operation="start"
            serverName="${serverName}" userDir="${wlp_usr}" outputDir="${wlp_output}" />
<wlp:server ref="wlp.ant.test" operation="status"/>
```

Liberty profile: Ant task - deploy:

The `deploy` task can be used to install applications on a Liberty profile server.

Description

The `deploy` task supports deployment of one or more applications to the Liberty profile server.

Attributes

The table below describes attributes of `deploy` task.

Table 2. The attributes of `deploy` task.

The first column contains a list of attributes, the second column contains a description of each attribute, and the third column represents whether this attribute is required.

Attribute	Description	Required
`installDir`	Location of the Liberty profile server.	Yes
`serverName`	Name of the Liberty profile server instance. The default value is `defaultServer`.	No
`userDir`	Value of the `${wlp_user_dir}` variable. The default value is `${installDir}/user/servers/${serverName}`.	No
`outputDir`	Value of the `${wlp_output_dir}` variable. The default value is `${installDir}/user/servers/${serverName}`.	No
`file`	Location of a single application to be deployed. See `file` attribute in Apache Ant. The application type can be `war`, `eba`, `zip`, `rar` or `jar`.	Yes, only when the `fileset` attribute is not specified.
`fileset`	Location of multiple applications to be deployed. See `fileset` attribute in Apache Ant.	Yes, only when the `file` attribute is not specified.
`timeout`	Waiting time before the deployment completes successfully. The default value is 30 seconds.	No
`ref`	Reference to an existing server task definition to reuse its server configuration. The value can be null when other required attributes are set.	No

Example

This example shows how to use the `deploy` task in your `build.xml` file:

```
<wlp:deploy ref="wlp.ant.test" >
    <fileset dir="${basedir}/resources/">
        <include name="**/*.war"/>
    </fileset>
</wlp:deploy>
<wlp:deploy ref="wlp.ant.test" file="${basedir}/resources/SimpleOSGiApp.eba"  timeout="40000"/>
```

Liberty profile: Ant task - undeploy:

The `undeploy` task can be used to remove applications from a Liberty profile server.

Description

The `undeploy` task supports undeployment of a single application from the Liberty profile server.

Attributes

The table below describes attributes of the `undeploy` task.

Table 3. The attributes of the `undeploy` task.

The first column contains a list of attributes, the second column contains a description of each attribute, and the third column represents whether this attribute is required.

Attribute	Description	Required
`installDir`	Location of the Liberty profile server.	Yes
`serverName`	Name of the Liberty profile server instance. The default value is `defaultServer`.	No
`userDir`	Value of the `${wlp_user_dir}` variable. The default value is `${installDir}/user/servers/${serverName}`.	No
`outputDir`	Value of the `${wlp_output_dir}` variable. The default value is `${installDir}/user/servers/${serverName}`.	No
`file`	Name of application to be removed. The application type can be `war`, `eba`, `zip`, `rar` or `jar`.	Yes
`timeout`	Waiting time before the undeployment completes successfully. The default value is 30 seconds.	No
`ref`	Reference to an existing server task definition to reuse its server configuration. The value can be null when other required attributes are set.	No

Example

This example shows how to use the `undeploy` task in your `build.xml` file:

```
<wlp:undeploy ref="wlp.ant.test" file="SimpleOSGiApp.eba" timeout="60000" />
```

Adding and removing server features

Features are the units of capability by which you control the pieces of the runtime environment that are loaded into a particular server. To add or remove a server feature, you add or remove an XML snippet in the `<feature>` sub-element of the `server.xml` configuration file. When you add or remove server features, the changes are applied dynamically.

Before you begin

You can add and remove server features as described in this topic, or as described in "Editing the Liberty profile configuration using developer tools" on page 126.

About this task

These are the XML snippets that enable the main features:

- `<feature>appSecurity-1.0</feature>`

- `<feature>beanvalidation-1.0</feature>`
- `<feature>blueprint-1.0</feature>`
- `<feature>jdbc-4.0</feature>`
- `<feature>jpa-2.0</feature>`
- `<feature>jsf-2.0</feature>`
- `<feature>jsp-2.2</feature>`
- `<feature>localConnector-1.0</feature>`
- `<feature>servlet-3.0</feature>`
- `<feature>ssl-1.0</feature>`
- `<feature>wab-1.0</feature>`

Including a feature in the configuration might cause one or more additional, required features to be loaded automatically. For example, if you include the `wab-1.0` feature, then the `servlet-3.0` and `blueprint-1.0` features are loaded automatically. For more information about these features, see Liberty profile: Server features.

Procedure

1. Open the `server.xml` configuration file for editing.

   You can do this using a text editor. By default, the path and filename for the configuration root document file is `usr/servers/`*server_name*`/server.xml`. However, you can change the path. See Specifying the location of the server.xml file.

2. Add or remove features in the configuration file.

   The set of features is enclosed within the `<featureManager>` element, and each feature within the `<feature>` sub-element. For example:

   ```
   <server>
       <featureManager>
           <feature>servlet-3.0</feature>
           <feature>localConnector-1.0</feature>
       </featureManager>
   </server>
   ```

   The matching of feature names is not case-sensitive. This is also a valid server configuration:

   ```
   <featureManager>
           <feature>Servlet-3.0</feature>
           <feature>localConnector-1.0</feature>
   </featureManager>
   ```

3. Save the changes to the configuration file.

Results

Your changes are applied. If the server is running, the changes are applied dynamically.

Using includes in the configuration

You can keep all your configuration in a single `server.xml` file, or you can use include elements to consolidate configurations from separate files to create the structure that is most useful to you as described in this topic.

About this task

It can be easier to maintain a complex configuration by splitting it across a set of files. For example:
- You might want to include a file that contains variables that are specific to the local host, so that your main configuration can be used on multiple hosts.

- You might want to keep all of the configuration for a particular application in a separate file that can be versioned with the application itself.

Example

This is the syntax for including a configuration file:

```
<include location="pathname/filename"/>
or
<include location="url"/>
```

The following list shows the possible locations; they are searched in the order shown.

1. in a location specified relative to the parent file
2. in the shared configuration directory
3. in a location specified as an absolute path
4. on a web server

To ensure that your include configuration behaves predictably, you need to be aware of the following processing rules for included configuration files:

- For singleton services such as logging, or application monitoring, entries are processed in the order they appear in the file and later entries add to or override previous ones. This is also true for configuration instances, for example an application or data source, where the configuration instances have the same ID.
- Include statements can be placed anywhere within the `<server />` element.
- Each included file must contain a `<server />` element that matches the one in the parent configuration file.
- Included files can nest other included files.
- Each included file is logically merged into the main configuration at the position that the <include /> statement occurs in the parent file.

In the following example, the primary server configuration file `server.xml` includes the contents of the `blogDS.xml` configuration file, which is located in the shared configuration directory. The `blogDS.xml` file contains a data source definition. This definition has been put in a separate configuration file so that it can be included in several different `server.xml` files, and thereby used across multiple server instances.

Here is example code from the `server.xml` file:

```
<server>
  <featureManager>
    <feature>servlet-3.0</feature>
    <feature>jdbc-4.0</feature>
  </featureManager>
  <application id="blog" location="blog.war" name="blog" type="war"/>
  <include location="${shared.config.dir}/blogDS.xml"/>
</server>
```

Here is the example code from the `blogDS.xml` file:

```
<server>
  <dataSource id="blogDS" jndiName="jdbc/blogDS" jdbcDriverRef="derbyEmbedded">
    <properties createDatabase="create" databaseName="C:/liberty/basics/derby/data/blogDB" />
  </dataSource>
  <jdbcDriver id="derbyEmbedded"/>
    <library>
      <fileset dir="C:/liberty/basics/derby" includes="derby.jar" />
    </library>
  </jdbcDriver>
</server>
```

Using variables in the configuration

You can use variables in the configuration to avoid hard coding values that might not be appropriate when the configuration is reused in different environments.

About this task

Variables can be defined by setting a property in any of the following places:
- in the server configuration file, or an included file
- in the `bootstrap.properties` file
- on the `ws-launch.jar` command script that is used to start the server

Best practice: Variables that are specific to a particular server, for example port numbers, are usually specified in the `bootstrap.properties` file, allowing the `server.xml` to be shared across multiple servers while keeping those values different in each server. Variables that are shared across a group of servers, for example database configuration for a particular host, is better specified in an xml file that is included into the parent configuration file.

Procedure
- Specify a variable in a configuration file.

  Variables defined in the configuration files are scoped to the configuration elements by which they are used. For example, the following code fragment creates a variable called *updateTrigger_var* to be used in **applicationMonitor** configuration elements:

  ```
  <applicationMonitor updateTrigger_var="mbean" />
  ```

  To create a variable that is used in a particular configuration instance (such as an application or resource entry), you must also specify the instance identifier. For example:

  ```
  <httpEndpoint id="defaultHttpEndpoint" HTTP_default_var="8889" />
  ```
- Specify a variable in the `bootstrap.properties` file.

  Variables defined in the `bootstrap.properties` file are not scoped to particular configuration elements. You enter the variables as key-value pairs. For example:

  ```
  HTTP_default_var=8006
  ```
- Specify a variable on the `ws-launch.jar` command script.

  Variables defined on the Java command are not scoped to particular configuration elements. You enter the variables as Java system properties. For example:

  ```
  java -DHTTP_default_var=8008 -jar ws-launch.jar myServer
  ```
- Use a defined variable in the configuration.

  The variable substitution syntax is ${*variable_name*}. For example, to use the *HTTP_default_var* variable, add the following code fragment to the configuration file:

  ```
  <httpEndpoint id="defaultHttpEndpoint"
  httpPort="${HTTP_default_var}">
  </httpEndpoint>
  ```
- Use `variable` element in the configuration

  You can use the `variable` element to define a variable globally in the server configuration. If the same variable is defined in an included file, it is overridden by the one in the `server.xml` file . For example, to use the `variable` element, add the following code fragment to the configuration file:

  ```
  <variable name="HTTP_default_var" value="8889" />
  ```
- Override inheritable attributes in the configuration

  You can override the default values of inheritable attributes in the configuration. The inheritable attributes are listed on the "Liberty profile: Configuration elements in the server.xml file" on page 3 page with an `Inherits` type. For example, the **onError** attribute is one of inheritable attributes. You can define a variable name for the **onError** attribute globally by either setting it in the `bootstrap.properties` or

server.xml file with a `variable` element. If the same variable name is specified in both files, the value in the `server.xml` file is used. If the attribute is not explicitly set in either of two files, it uses the default value. If an invalid value is set to the inheritable attribute, the attribute value falls back to the global value defined in `bootstrap.properties` or `server.xml` file or the default value if not defined at the global level.

Another example is logging properties in the Liberty profile. See Liberty profile: Trace and logging.

Controlling dynamic updates

There are three types of dynamic update that can be controlled through configuration: changing the server configuration; adding and removing applications; updating installed applications. For all deployed applications, you can configure if application monitoring is enabled, and how often to check for updates to applications. For the "dropins" directory, you can also configure the name and location of the directory, and choose whether or not to deploy the applications that are in the directory.

About this task

By default, deployed applications are monitored for updates, and the updates are dynamically applied to the running application. This applies both to applications that are deployed through configuration entries, and those deployed from the "dropins" directory. You can change these default behaviors by setting the `config` and `applicationMonitor` elements in the `server.xml` configuration file. You can use a text editor to do this, or you can use the developer tools and select **Configuration Admin Service** or **Application Monitor** in the server configuration design view.

See also the descriptions of the `config` and `applicationMonitor` elements in "Liberty profile: Configuration elements in the server.xml file" on page 3.

Note: The default settings for application monitoring are as follows:

```
<applicationMonitor updateTrigger="polled" pollingRate="500ms"
                    dropins="dropins" dropinsEnabled="true"/>
```

Procedure

- Configure dynamic changes to the server configuration.

  Changes to the `server.xml` file, or any files it includes, are detected by the runtime environment and applied to the active configuration. You can disable this behavior by setting the **config** element in the `server.xml` file:

  ```
  <config updateTrigger="disabled"/>
  ```

  Note: The **updateTrigger** property has three possible values:

  polled   The runtime environment scans the `server.xml` file for changes using the timing interval specified by the **monitorInterval** property.

  mbean
  The runtime environment only looks for updates when prompted to do so through a call to an mbean. This is the mode that is used by the developer tools to update the `server.xml` file, unless you override it.

  disabled
  The updates are not dynamically applied.

  When you specify the polling rate, you include the unit of time after the number:
  - `ms` (milliseconds)
  - `s` (seconds)
  - `m` (minutes)
  - `h` (hours)

You cannot mix these units of time, and you can only use integer values. For example, for a 1.5 second polling rate you specify `1500ms`; you cannot specify `1s500ms` or `1.5s`.

- Configure dynamic addition and removal of applications.

  As described in Deploying applications to the Liberty profile, applications can be dynamically added to and removed from the server runtime through two mechanisms:

  - By adding or removing application entries in the `server.xml` file.

    If you disable dynamic changes to the server configuration as described in the previous step, then adding or removing application entries has no effect on a running server. Your changes are only applied at the next server restart. Note that the changes are picked up immediately, if you update application entries using the developer tools.

  - By moving application files into and out of the "dropins" directory.

    This behavior can be controlled by setting the **applicationMonitor** element in the `server.xml` file. For example, to disable dynamic installation of applications from the "dropins" location, create an entry like this:

    ```
    <applicationMonitor dropinsEnabled="false"/>
    ```

    Note: The **dropinsEnabled** property is a boolean property that determines whether or not the applications in the "dropins" directory are deployed.

- Configure dynamic updates to installed applications.

  By default, if you add, remove or modify any files within a deployed application, or you replace the whole application with an updated version, the previous version is automatically stopped and the new version is started. This process applies for any deployed application, whether the application is located in the "dropins" directory or at a location defined in the `server.xml` file. You can control this behavior by setting the **applicationMonitor** element in the `server.xml` file. For example, to disable dynamic update of all applications, create an entry like this:

  ```
  <applicationMonitor updateTrigger="disabled"/>
  ```

  Note: The **updateTrigger** property has three possible values:

  polled The runtime environment scans the deployed application files for changes using the timing interval specified by the **pollingRate** property.

  mbean

  The runtime environment only looks for updates when prompted to do so through a call to an mbean. This is the mode that is used by the developer tools to update applications, unless you override it.

  disabled

  Application updates are not dynamically applied. To update a deployed application when this mode is active, uninstall then reinstall the application.

  When you specify the polling rate, you include the unit of time after the number:

  - `ms` (milliseconds)
  - `s` (seconds)
  - `m` (minutes)
  - `h` (hours)

  You cannot mix these units of time, and you can only use integer values. For example, for a 1.5 second polling rate you specify `1500ms`; you cannot specify `1s500ms` or `1.5s`.

- Configure the name and location of the "dropins" directory.

  By default, the "dropins" directory is `${server.config.dir}/apps`. You can change this by setting the **applicationMonitor** element in the `server.xml` file. For the location, you can use any known variable, or a property in the `bootstrapping.properties` file, or an absolute path, or a path relative to the server directory. For example, both the following settings point to the same location:

```
<applicationMonitor dropins="${server.config.dir}/applications" />
<applicationMonitor dropins="applications" />
```

Liberty profile: Using Ref tags in configuration files

Ref tags make it possible to define a common configuration element, then reuse that definition by referring to it (using a Ref tag) from elsewhere in the configuration. Ref tags can be used in the same configuration file that contains the element definition, or in an included configuration file.

Different approaches are used to specify relationships between the required configuration elements. For example, the following data source definitions are all valid. The first uses no Ref tags, the second uses a combination of direct element definition and Ref tags, and the third uses Ref tags only.

Example 1: Using no Ref tags

```
<dataSource id="blogDS" jndiName="jdbc/blogDS">
  <properties createDatabase="create" databaseName="C:/liberty/basics/derby/data/blogDB"/>
  <jdbcDriver>
    <library>
      <fileset dir="C:/liberty/basics/derby" includes="derby.jar"/>
    </library>
  </jdbcDriver>
  <connectionManager maxPoolSize="10"/>
</dataSource>
```

Example 2: Combining direct element definition and Ref tags

```
<dataSource id="blogDS" jndiName="jdbc/blogDS" connectionManagerRef="derbyPool">
  <properties createDatabase="create" databaseName="C:/liberty/basics/derby/data/blogDB"/>
  <jdbcDriver libaryRef="derbyLib"/>
</dataSource>

<connectionManager id="derbyPool" maxPoolSize="10"/>

<library id="derbyLib"/>
  <fileset dir="C:/liberty/basics/derby" includes="derby.jar"/>
</library>
```

Example 3: Using Ref tags only (except for the `properties` element, which is only permitted as nested)

```
<dataSource id="blogDS" jndiName="jdbc/blogDS"
            connectionManagerRef="derbyPool" jdbcDriverRef="derbyEmbedded">
  <properties createDatabase="create" databaseName="C:/liberty/basics/derby/data/blogDB"/>
</dataSource>

<connectionManager id="derbyPool" maxPoolSize="10"/>

<jdbcDriver id="derbyEmbedded" libraryRef="derbyLib"/>

<library id="derbyLib" filesetRef="derbyFileset"/>

<fileset id="derbyFileset" dir="C:/liberty/basics/derby" includes="derby.jar"/>
```

Configuring the Liberty profile with a web server plug-in

You can configure a web server plug-in so that, when the web server receives an HTTP request for dynamic resources, the request is forwarded to the Liberty profile.

About this task

A web server plug-in is used to forward HTTP requests from a supported web server to one or more application servers. The plug-in takes a request and checks the request against configuration data in the

`plugin-cfg.xml` file. The configuration data maps the URI for the HTTP request to the hostname of an application server. The web server plug-in then uses this information to forward the request to the application server.

Procedure

1. Start the server that hosts your applications, and ensure that the `localConnector-1.0` feature and other required features are included in the server configuration.

   In the `pluginConfiguration` element of the server configuration file, you can specify the **webserverPort** and **webserverSecurePort** attributes to forward requests from the web server. By default, the value of **webserverPort** is 80 and the value of **webserverSecurePort** is 443. However, you might want to change these settings. For example for Linux and similar platforms, if you are a non-root user, you must use port numbers greater than 1024.

   Here is an example of a `server.xml` server configuration file:

   ```
   <server description="new server">
    <featureManager>
      <feature>localConnector-1.0</feature>
      <feature>jsp-2.2</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080">
           <tcpOptions soReuseAddr="true" />
       </httpEndpoint>

    <pluginConfiguration webserverPort="8088" webserverSecurePort="8443"/>

    <application type="war" id="myapp" name="myapp" location="${server.config.dir}/apps/myapp.war" />
    <application type="war" id="snoop" name="snoop" location="${server.config.dir}/apps/snoop.war" />
   </server>
   ```

2. Generate the `plugin-cfg.xml` file for your Liberty profile server and applications.

   a. Using the same JDK as the server, run the **jconsole** Java utility in a command window.

      For example, run the following command:

      ```
      c:\java\bin\jconsole
      ```

      The server process should be listed in the choices waiting for connection.

   b. Connect to your server then click the **mbeans** tab for the `websphere` mbeans.

   c. On the **mbeans** tab, select and invoke the **defaultPluginConfig generation** mbean operation to generate the `plugin-cfg.xml` file.

   Here is an example of a `plugin-cfg.xml` file:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <Config ASDisableNagle="false" AcceptAllContent="false" AppServerPortPreference="HostHeader"
           ChunkedResponse="false" FIPSEnable="false" IISDisableNagle="false"
           IISPluginPriority="High" IgnoreDNSFailures="false" RefreshInterval="60"
           ResponseChunkSize="64" SSLConsolidate="false" SSLPKCSDriver="REPLACE"
           SSLPKCSPassword="REPLACE" TrustedProxyEnable="false" VHostMatchingCompat="false">
      <Log LogLevel="Error" Name=".\logs\defaultServer\http_plugin.log"/>
      <Property Name="ESIEnable" Value="true"/>
      <Property Name="ESIMaxCacheSize" Value="1024"/>
      <Property Name="ESIInvalidationMonitor" Value="false"/>
      <Property Name="ESIEnableToPassCookies" Value="false"/>
      <Property Name="PluginInstallRoot" Value="."/>
      <VirtualHostGroup Name="default_host">
            <VirtualHost Name="*:8088"/>
            <VirtualHost Name="*:8443"/>
            <VirtualHost Name="*:9080"/>
        </VirtualHostGroup>
      <ServerCluster CloneSeparatorChange="false" GetDWLMTable="false" IgnoreAffinityRequests="true"
   ```

```
                LoadBalance="Round Robin" Name="defaultServer_default_node_Cluster"
                PostBufferSize="64" PostSizeLimit="-1" RemoveSpecialHeaders="true"
                RetryInterval="60">
        <Server CloneID="b564bdc7-2c27-4a4b-ad37-9213c66e60d1" ConnectTimeout="0"
                ExtendedHandshake="false" MaxConnections="-1" Name="default_node_defaultServer0"
                ServerIOTimeout="900" WaitForContinue="false">
          <Transport Hostname="somehost.example.com" Port="9080" Protocol="http"/>
        </Server>
      <PrimaryServers>
          <Server Name="default_node_defaultServer0"/>
        </PrimaryServers>
    </ServerCluster>
  <UriGroup Name="default_host_defaultServer_default_node_Cluster_URIs">
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/myapp/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/snoop/*"/>
    </UriGroup>
  <Route ServerCluster="defaultServer_default_node_Cluster"
        UriGroup="default_host_defaultServer_default_node_Cluster_URIs"
        VirtualHostGroup="default_host"/>
</Config>
```

The `plugin-cfg.xml` file is generated in the `${server.output.dir}` directory.

3. Copy the `plugin-cfg.xml` file to the machine that hosts the web server.

4. Install the `plugin-cfg.xml` file on the web server, then configure the web server.

   Typically, you have to enable the plug-in within the `http.conf` file of the web server by using the `LoadModule` phrase, and specify the location of `plugin-cfg.xml` file using the `WebSpherePluginConfig` phrase. For example:

   - On Windows systems: **Windows**

     ```
     LoadModule was_ap22_module "path\to\mod_was_ap22_http.dll"
     WebSpherePluginConfig "path\to\plugin-cfg.xml"
     ```

   - On other distributed systems: **AIX** **Linux** ▶ **AIX** ▶ **HP-UX** ▶ **Solaris** **HP-UX** **Solaris**

     ```
     LoadModule was_ap22_module "path/to/mod_was_ap22_http.so"
     WebSpherePluginConfig "path/to/plugin-cfg.xml"
     ```

5. Optional: If you want the web server plug-in to forward HTTP requests to more than one Liberty profile server, repeat the previous steps for each additional server. Make sure you consolidate all the plug-in configurations into one `plugin-cfg.xml` file.

   Note: If an application programmatically modifies the session cookie configuration using Servlet 3.0 APIs, then the application needs to be initialized prior to generating the `plugin-cfg.xml` file. Otherwise, the **AffinityCookie** attribute defined for that application might be wrong. To avoid this problem, you can set **deferServletLoad** to false , start the server, generate the plugin, and then remove the **deferServletLoad** attribute

Configuring session persistence for the Liberty profile

When session data must be maintained across a server restart or an unexpected server failure, you can configure the Liberty profile to persist the session data to a database. This configuration allows multiple servers to share the same session data, and session data can be recovered in the event of a failover.

About this task

To configure one or more servers in the Liberty profile to persist session data to a database, complete the following steps.

Procedure

1. Define a shared session management configuration that you can reuse among all of your servers. You must at least:

a. Enable the `sessionDatabase-1.0` feature.

b. Define a data source:

```
<dataSource id="SessionDS" ... />
```

c. Reference the data source from the session database configuration.

```
<httpSessionDatabase id="SessionDB" dataSourceRef="SessionDS" ... />
```

d. Reference the persistent storage location from the session management configuration.

```
<httpSession storageRef="SessionDB" ... />
```

See "Liberty profile: Configuration elements in the server.xml file" on page 3 for details about the `httpSession` and `httpSessionDatabase` elements.

For example, you can create a file named `${shared.config.dir}/httpSessionPersistence.xml` as follows:

```
<server description="Demonstrates HTTP Session Persistence Configuration">

    <featureManager>
        <feature>sessionDatabase-1.0</feature>
        <feature>servlet-3.0</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="${httpPort}">
        <tcpOptions soReuseAddr="true"/>
    </httpEndpoint>

    <fileset id="DerbyFiles" includes="*.jar" dir="${shared.resource.dir}/derby/client"/>
    <library id="DerbyLib" filesetRef="DerbyFiles"/>
    <jdbcDriver id="DerbyDriver" libraryRef="DerbyLib"/>
    <dataSource id="SessionDS" jdbcDriverRef="DerbyDriver" jndiName="jdbc/sessions">
        <properties.derby.client user="user1" password="password1"
                                 databaseName="${shared.resource.dir}/databases/SessionDB"
                                 createDatabase="create"/>
    </dataSource>

    <httpSessionDatabase id="SessionDB" dataSourceRef="SessionDS"/>
    <httpSession storageRef="SessionDB" cloneId="${cloneId}"/>

    <application id="test" name="test" type="ear" location="${shared.app.dir}/test.ear"/>

</server>
```

Note: When multiple servers are configured to persist session data to the same database, those servers must share the same session management configuration. Any other configuration is not supported. For example, it is not possible for one server to use a multi-row schema while another server uses a single-row schema.

Best Practice: If session affinity is important to your application, explicitly define a unique clone ID for each server. In that way, you don't have to depend on the default clone ID generated by the server, and you can be certain that the value of the clone ID never changes.

2. Include the shared session management configuration in each of your servers. For example, create two `server.xml` files for server instances named `s1` and `s2`, as follows:

- `${wlp.user.dir}/servers/s1/server.xml`

- `${wlp.user.dir}/servers/s2/server.xml`

```
<server description="Example Server">
    <include location="${shared.config.dir}/httpSessionPersistence.xml"/>
</server>
```

3. Specify unique variables in the `bootstrap.properties` file of each server.

- `${wlp.user.dir}/servers/s1/bootstrap.properties`

```
httpPort=9081
cloneId=s1
```
- ${wlp.user.dir}/servers/s2/bootstrap.properties
```
httpPort=9082
cloneId=s2
```
4. You might have to create a table for session persistence before you start the servers.

   a. **Optional:** If you want to change the default row size, table name, or table space name, see "Liberty profile: Configuration elements in the server.xml file" on page 3 for details about the `httpSessionDatabase` element.

   b. No additional action is required if your server is installed on one of the distributed operating systems. The server automatically creates the table.

   c. **Optional:** If your server is using DB2 for session persistence, you can increase the page size to optimize performance for writing large amounts of data to the database.

5. **Optional:** You can integrate HTTP sessions and security in the Liberty profile. By default, after a session is created and accessed within a protected resource with security enabled, only the originating owner of that session can access it. Session security (security integration) is enabled by default.

6. **Optional:** Install and configure the web server plug-in to route requests to each of the servers you configured. Note that session affinity is only maintained if your plug-in configuration specifies clone IDs that match the clone IDs defined in the server configuration.

Accessing JMX connectors on the Liberty profile

This topic describes how to access Java Management Extensions (JMX) connectors on the Liberty profile.

About this task

There are two JMX connectors supported on the Liberty profile, each connector is enabled through a different server feature: `localConnector-1.0` and `restConnector-1.0`.

- The local connector is enabled through the server feature `localConnector-1.0`. Access through the local connector is protected by the policy implemented by the SDK in use. Currently the SDKs require that the client runs on the same host as the Liberty profile, and under the same user ID.

- The REST connector is enabled through the server feature `restConnector-1.0`. Remote access through the REST connector is protected by a single administrator role. In addition, SSL is required to keep the communication confidential. The `restConnector-1.0` feature already includes the `ssl-1.0` feature.

Note: An application deployed on the Liberty profile has unrestricted access to its MBeanServer directory.

The following section describes how to configure and access the REST connector on the Liberty profile.

Procedure

1. Enable the REST connector using the following code in the `server.xml` file.
```
<featureManager>
    <feature>restConnector-1.0</feature>
</featureManager>
```
2. Configure SSL certificates in the `server.xml` file.

3. Configure a user or group to the administrator role in the `server.xml` file.
   - Map to the administrator role for the Liberty profile

4. Access the REST connector from a JMX client application or using the `jConsole` tool provided in the Java SDK. Use `-J` flags to pass the system properties as Java options and set the class path to include the connector class files. The connector class files are packed in the `clients/restConnector.jar` file.
   - Use the following properties for SSL certificates:

```
-J-Djavax.net.ssl.trustStore=<location of your client trust store>
-J-Djavax.net.ssl.trustStorePassword=<password for the trust store>
-J-Djavax.net.ssl.trustStoreType=<type of trustore>
```

An example of using the `jConsole` tool with SSL configurations is as follows:

```
jconsole -J-Djava.class.path=%JAVA_HOME%/lib/jconsole.jar;
                    %JAVA_HOME%/lib/tools.jar;
                    %WLP_HOME%/clients/restConnector.jar
         -J-Djavax.net.ssl.trustStore=key.jks
         -J-Djavax.net.ssl.trustStorePassword=Liberty
         -J-Djavax.net.ssl.trustStoreType=jks
```

After the `jConsole` starts, select **Remote Process**, and enter the JMX service URL:
`service:jmx:rest://<host>:<port>/IBMJMXConnectorREST`. The port number is the HTTPS port. You
must provide the username and password as well.

Note:

> There are some JMX REST connection options that you can specify as system properties, see
> "Liberty profile: JMX REST connector settings" on page 150.

Mapping the administrator role for the Liberty profile

You can use `quickStartSecurity` element or any supported user registries for the administrator role
mapping on the Liberty profile.

About this task

All the JMX methods and MBeans accessed through the REST connector are currently protected by a
single role named "administrator". To get started quickly, use `quickStartSecurity` element to configure a
single user with administrator role and configure the default SSL configuration.

You can also use any supported user registry. You can not use `quickStartSecurity` element if you have
already configured another user registry. In this case, you have to map users or roles from the registry to
the administrator role.

Procedure

- Use `quickStartSecurity` element for a single user mapping.

  Here is an example showing the minimal required configuration:

  ```
  <featureManager>
     <feature>restConnector-1.0</feature>
  </featureManager>
  <quickStartSecurity userName="bob"  userPassword="bobpassword" />
  <keyStore id="defaultKeyStore" password="keystorePassword"/>
  ```
- Or use the basic registry for administrator role mapping.

  Here is an example using the basic registry giving the user "bob" or the group "group1" administrator
  role:

  ```
  <basicRegistry>
      <user name="bob" password="bobpassword"/>
      <user name="joe" password="joepassword"/>
      <group name="group1" ...>
      </group>
  </basicRegistry>

  <administratorRole>
      <user>bob</user>
      <group>group1</group>
  </administratorRole>
  ```

- Or use the LDAP registry for administrator role mapping.

  Here is an example using the LDAP registry giving the user "bob" administrator role.

  ```
  <ldapRegistry id="basic" host="" port="">
      <tds.properties ... />
  </ldapRegistry>

  <adminstratorRole>
      <user>cn=bob,o=ibm,c=us</user>
  </administratorRole>
  ```

Developing a JMX Java client for the Liberty profile

This topic describes how to develop a Java Management Extensions (JMX) client application and then you can access the secured REST connector on the Liberty profile.

About this task

Using a JMX remote client application, you can administer the Liberty profile through JMX programming.

Procedure

- Develop a sample JMX client as follows. The REST connector supports the standard JMX API.

  ```
  import javax.management.remote.JMXServiceURL;
  import javax.management.MBeanServerConnection;
  import javax.management.remote.JMXConnector;
  import javax.management.remote.JMXConnectorFactory;
  import java.util.HashMap;


  public class Test {

    public static void main(String[] args) {
      try {
        HashMap<String, Object> environment = new HashMap<String, Object>();
        environment.put("jmx.remote.protocol.provider.pkgs", "com.ibm.ws.jmx.connector.client");
        environment.put(JMXConnector.CREDENTIALS, new String[] { "bob", "bobpassword" });

        JMXServiceURL url = new JMXServiceURL("service:jmx:rest://<host>:<port>/IBMJMXConnectorREST");
        JMXConnector connector = JMXConnectorFactory.newJMXConnector(url, environment);
        connector.connect();
        MBeanServerConnection mbsc = connector.getMBeanServerConnection();
      } catch(Throwable t) {
          ...
      }
    }
  }
  ```

- Optional: Disable hostname verification for SSL certificates. The certificates installed with the Liberty profile might not contain the hostname of where the server is actually running. If you want to disable hostname verification of SSL certificates, you may set the system property `com.ibm.ws.jmx.connector.client.disableURLHostnameVerification` to true, which disables hostname verification for all connections. To disable hostname verification on a per-connection basis, pass the property as a new `environment` when creating the JMX connection:

  ```
  HashMap<String, Object> environment = new HashMap<String, Object>();
  environment.put("jmx.remote.protocol.provider.pkgs", "com.ibm.ws.jmx.connector.client");
  environment.put("com.ibm.ws.jmx.connector.client.disableURLHostnameVerification", Boolean.TRUE);
  environment.put(JMXConnector.CREDENTIALS, new String[] { "bob", "bobpassword" });
  ...
  ```

- Optional: Configure JMX REST connector settings by using the environment Map.

```
...
HashMap<String, Object> environment = new HashMap<String, Object>();
environment.put("com.ibm.ws.jmx.connector.client.rest.maxServerWaitTime", 0);
environment.put("com.ibm.ws.jmx.connector.client.rest.notificationDeliveryInterval", 65000);
...
```

Liberty profile: JMX REST connector settings

This topic provides a list of keys and description that you can use as system properties when connecting to a Liberty profile with JMX REST connectors. These keys are constants in the com.ibm.ws.jmx.connector.client.rest.ClientProvider class, and each requires an integer value that specifies an amount of time in milliseconds.

MAX_SERVER_WAIT_TIME

Amount of time that the client waits for the server to become available before the JMX connection fails and a new connection must be created. The key for constant **MAX_SERVER_WAIT_TIME** is the String `com.ibm.ws.jmx.connector.client.rest.maxServerWaitTime`. If the connection is restored, any previous notification listeners are registered again. To disable this behavior, set the value to `zero`.

NOTIFICATION_DELIVERY_INTERVAL

Maximum amount of time that the server waits for new notifications before responding to a request for notifications from the client. The key for constant **NOTIFICATION_DELIVERY_INTERVAL** is the String `com.ibm.ws.jmx.connector.client.rest.notificationDeliveryInterval`. A larger value results in better notification delivery times because less time is spent establishing new connections. Normally it is not necessary to adjust this value.

NOTIFICATION_INBOX_EXPIRY

Amount of time that the server waits before discarding notification registrations if the client has not checked for new notifications. The key for constant **NOTIFICATION_INBOX_EXPIRY** is the String `com.ibm.ws.jmx.connector.client.rest.notificationInboxExpiry`. Normally it is not necessary to adjust this value.

NOTIFICATION_READ_TIMEOUT

Read timeout for notification fetching. Because the server might wait up to **NOTIFICATION_DELIVERY_INTERVAL** before responding, this value must be somewhat larger, though normally it is not necessary to adjust this value. The key for constant **NOTIFICATION_READ_TIMEOUT** is the String `com.ibm.ws.jmx.connector.client.rest.notificationReadTimeout`.

READ_TIMEOUT

Read timeout for all client communications with the server, except notification fetching. Adjust this value if the client throws read timeout exceptions because of a slow connection or client or server process. The key for constant **READ_TIMEOUT** is the String `com.ibm.ws.jmx.connector.client.rest.readTimeout`.

SERVER_STATUS_POLLING_INTERVAL

Amount of time that the client waits between checks that the server is available again when **MAX_SERVER_WAIT_TIME** is non-zero. Normally it is not necessary to adjust this value. The key for constant **SERVER_STATUS_POLLING_INTERVAL** is the String `com.ibm.ws.jmx.connector.client.rest.serverStatusPollingInterval`.

You can enable those JMX REST connector options using the system properties. Take the **MAX_SERVER_WAIT_TIME** option as an example, you can use one of the following options:

- In the command prompt, set the system property
  `-Dcom.ibm.ws.jmx.connector.client.rest.maxServerWaitTime=0`
- In a JMX client program, add the setting
  `environment.put("com.ibm.ws.jmx.connector.client.rest.maxServerWaitTime", 0);`

Generating a Liberty profile server dump using developer tools

Using the Liberty profile Utilities menu, you can generate a server dump for support.

About this task

You can use the Utilities menu of your Liberty profile to generate a server dump to capture all service-related information for support.

Procedure

1. In the Servers view, right-click your Liberty server profile, and select **Utilities** > **Generate dump for support....**
2. In the **Generate Dump** page, click **Finish** to create the server dump file.
3. In the Console view, the path to the server dump is indicated:

   ```
   Server TestServer dump complete in
   D:\LibertyUnzip\wlp\usr\servers\TestServer\TestServer.dump-12.03.22_15.00.11.zip.
   ```

Generating a Liberty profile server dump from the command prompt

From the command prompt you can use the `server dump` command to capture status information for a Liberty profile server.

About this task

The `server dump` command is useful for problem diagnosis of a Liberty profile server because the result file contains server configuration, log information, and details of the deployed applications in the `workarea` directory. The command can be applied to either a running or a stopped server.

For a running server, the following information is also included:

- State of each OSGi bundle in the server
- Wiring information for each OSGi bundle in the server
- Component list managed by the Service Component Runtime (SCR)
- Detailed information of each component from SCR
- Configuration administration data of each OSGi bundle
- Information on registered OSGi services
- Runtime environment settings such as JVM, heap size, operating system, thread information, and network status

Procedure

1. Open a command prompt, then change directory to the `wlp` directory.
2. Capture the status information using the command line tool. If you do not specify a server name, `defaultServer` is used.
   - To create a snapshot of the server status, use `server dump` command.
     –
       ```
       bin/server dump server_name --archive=package_file_name.dump.zip --include=all
       ```

       where *package_file_name*`.dump.zip` is a filename that you choose. This filename can include a full path name. If the full path is omitted, a compressed file called *package_file_name*`.dump.zip` is created in the ${server.output.dir} directory.

     You can also use the `--include` option with this command. For example, `--include=all` option packages all the files under the server installation directory; `--include=use` option packages files in the ${WLP_USER_DIR} directory.

Note: The result file is created using UTF-8 encoding for entry names, so you must use a tool capable of opening the file using UTF-8 encoding for entry names. The `jar` command in a Java SDK uses this format.

Results

If the specified server does not exist, the command does not succeed. If the specified server exists, a result file is created that contains the status information of the server.

Packaging a Liberty profile server using developer tools

You can create a compressed file containing a server runtime environment, server configuration, and applications using the packaging wizard.

About this task

Because a Liberty server is lightweight, you might find it useful to package up your applications and server in a compressed file. You can then store this package, distribute it to colleagues, use it to deploy the application to a different location or to another machine, or even embed it in your product distribution.

Procedure

1. In the Servers view, Stop the server.
2. Right-click your Liberty server profile, and select **Utilities** > **Package Server...**.
3. In the Package Server page, in the **Archive** field, type a filename and path for your archive package, or click **Browse** to locate a filename and path. This filename can include a full path name. If the full path is omitted, a compressed file called *package_file_name*.zip is created in the ${server.output.dir} directory. In the **Include:** field, select whether to include:
   - **All server content**, which includes binary files.
   - **Only server configuration and applications.**
4. Click **Finish**.

Results

If the specified server does not exist, the package creation does not succeed. If the specified server exists, a compressed file is created that contains your applications and server.

Packaging a Liberty profile server from the command prompt

From the command prompt you can create a compressed file containing a server runtime environment, server configuration, and applications.

About this task

Because a Liberty server is lightweight, you might find it useful to package up your applications and server in a compressed file. You can then store this package, distribute it to colleagues, use it to deploy the application to a different location or to another machine, or even embed it in your product distribution.

Note: The result file is created using UTF-8 encoding for entry names, so you must use a tool capable of opening the file using UTF-8 encoding for entry names. The `jar` command in a Java SDK uses this format.

Procedure

1. Open a command prompt, then change directory to the `wlp` directory.
2. Stop the server.

3. Package the server.

   Run the following command. If you do not specify a server name, `defaultServer` is used. If you do not specify the **--archive** parameter, the value of *server_name* is used for *package_file_name*, and the compressed file is created in the ${server.output.dir} directory.

   -
     ```
     bin/server package server_name --archive=package_file_name.zip --include=all
     ```

     where *package_file_name*.zip is a filename that you choose. This filename can include a full path name. If the full path is omitted, a compressed file called *package_file_name*.zip is created in the ${server.output.dir} directory.

   You can also use the **--include** option with this command. For example, --include=all option packages all the files under the server installation directory; --include=use option packages files in the ${WLP_USER_DIR} directory.

Results

If the specified server does not exist, the command does not succeed. If the specified server exists, a compressed file is created that contains your applications and server.

Using an OSGi console

Eclipse Equinox currently provides an OSGi console that you can use to aid with debugging. This console is not available by default. When you work with the Liberty profile, you can enable this console by configuring a port for it.

About this task

The Liberty profile uses the Eclipse Equinox implementation of the OSGi core specification. Equinox currently provides an OSGi console. To enable this console, you first allocate a specific port to it by setting the `osgi.console` property in the `bootstrap.properties` file. Then you can use Telnet to connect to the console on that port, and explore the OSGi framework.

Procedure

- Allocate a specific port to the OSGi console.

  You set the OSGi console port by specifying the `osgi.console` property. You set this property as a bootstrap property in the `bootstrap.properties` file. See Specifying Liberty profile bootstrap properties.

  The OSGi console is disabled when the `osgi.console` property is not set.
- Use Telnet to connect to the OSGi console port.
- Use the console to explore the framework.

  The available commands vary, depending on the OSGi framework being used. Command-line help provides enough information to get started.

Administering data access applications on the Liberty profile

The Liberty profile provides support to the data access applications using server features such as `jdbc-4.0` and other features.

Procedure

- Configure database connectivity on the Liberty profile.
- Configure connection pooling for database connections
- Configure database transaction recovery

Configuring database connectivity in the Liberty profile

You can configure a data source associated with different JDBC providers for database connectivity. The JDBC providers supply the driver implementation classes that are required for JDBC connectivity with your specific vendor database.

About this task

To access a database from your application, you must use a data source. Data sources are provided by JDBC drivers and come in the following varieties:

- `javax.sql.DataSource`

  This is the basic form of a data source. It does not provide interoperability that enhances connection pooling, and cannot participate as a two-phase capable resource in transactions involving multiple resources.

- `javax.sql.ConnectionPoolDataSource`

  This type of data source is enabled for connection pooling. It cannot participate as a two-phase capable resource in transactions involving multiple resources.

- `javax.sql.XADataSource`

  This type of data source is both enabled for connection pooling and is able to participate as a two-phase capable resource in transactions involving multiple resources.

In order to be usable in the Liberty profile, your JDBC driver must provide at least one of these types of data sources. For the commonly used JDBC drivers, the Liberty profile is already aware of the implementation class names for the various data source types. You only need to tell the Liberty profile where to find the JDBC driver.

Procedure

1. In the `server.xml` file, define a shared library pointing to the location of your JDBC driver JAR or compressed files. For example:

```
<library id="DB2JCC4Lib">
    <fileset dir="C:/DB2/java" includes="db2jcc4.jar db2jcc_license_cisuz.jar"/>
</library>
```

2. Define a data source using the JDBC driver. If you don't specify the type of data source, the Liberty profile chooses in the following order depending on which is available.

   - `javax.sql.ConnectionPoolDataSource`
   - `javax.sql.DataSource`
   - `javax.sql.XADataSource`

   Here is an example that accepts the default for data source type:

```
<dataSource id="db2" jndiName="jdbc/db2">
    <jdbcDriver libraryRef="DB2JCC4Lib"/>
    <properties.db2.jcc databaseName="SAMPLEDB" serverName="localhost" portNumber="50000"/>
</dataSource>
```

   Here is an example that uses `javax.sql.XADataSource` type:

```
<dataSource id="db2xa" jndiName="jdbc/db2xa" type="javax.sql.XADataSource">
    <jdbcDriver libraryRef="DB2JCC4Lib"/>
    <properties.db2.jcc databaseName="SAMPLEDB" serverName="localhost" portNumber="50000"/>
</dataSource>
```

3. Optional: Configure attributes for the data source, such as JDBC vendor properties and connection pooling properties.

   For example:

```
<dataSource id="db2" jndiName="jdbc/db2" connectionSharing="MatchCurrentState"
            isolationLevel="TRANSACTION_READ_COMMITTED" statementCacheSize="20">
    <connectionManager maxPoolSize="20" minPoolSize="5"
```

```
                    connectionTimeout="10s" agedTimeout="30m"/>
    <jdbcDriver libraryRef="DB2JCC4Lib"/>
    <properties.db2.jcc databaseName="SAMPLEDB" serverName="localhost" portNumber="50000"
                    currentLockTimeout="30s" user="user1" password="pwd1"/>
</dataSource>
```

For a full list of configuration attributes for the `dataSource` element, `connectionManager` element and some commonly used JDBC vendors, see "Liberty profile: Configuration elements in the server.xml file" on page 3.

4. Optional: Configure data sources for commonly used databases according to the following examples.

For DB2

```
<dataSource id="db2" jndiName="jdbc/db2">
    <jdbcDriver libraryRef="DB2JCC4Lib"/>
    <properties.db2.jcc databaseName="SAMPLEDB" serverName="localhost" portNumber="50000"/>
</dataSource>

<library id="DB2JCC4Lib">
    <fileset dir="C:/DB2/java" includes="db2jcc4.jar db2jcc_license_cisuz.jar"/>
</library>
```

For DB2 on iSeries® (Native)

```
<dataSource id="db2iNative" jndiName="jdbc/db2iNative">
    <jdbcDriver libraryRef="DB2iNativeLib"/>
    <properties.db2.i.native databaseName="*LOCAL"/>
</dataSource>

<library id="DB2iNativeLib">
    <fileset dir="/QIBM/Proddata/java400/jdk6/lib/ext" includes="db2_classes16.jar"/>
</library>
```

For DB2 on iSeries (Toolbox)

```
<dataSource id="db2iToolbox" jndiName="jdbc/db2iToolbox">
    <jdbcDriver libraryRef="DB2iToolboxLib"/>
    <properties.db2.i.toolbox databaseName="SAMPLEDB" serverName="localhost"/>
</dataSource>

<library id="DB2iToolboxLib">
    <fileset dir="/QIBM/ProdData/Http/Public/jt400/lib" includes="jt400.jar"/>
</library>
```

For Derby Embedded

```
<dataSource id="derbyEmbedded" jndiName="jdbc/derbyEmbedded">
    <jdbcDriver libraryRef="DerbyLib"/>
    <properties.derby.embedded databaseName="C:/databases/SAMPLEDB" createDatabase="create"/>
</dataSource>

<library id="DerbyLib">
    <fileset dir="C:/db-derby-10.8.1.2-bin/lib"/>
</library>
```

For Derby Network Client

```
<dataSource id="derbyClient" jndiName="jdbc/derbyClient">
    <jdbcDriver libraryRef="DerbyLib"/>
    <properties.derby.client databaseName="C:/databases/SAMPLEDB" createDatabase="create"
                    serverName="localhost" portNumber="1527"/>
</dataSource>

<library id="DerbyLib">
    <fileset dir="C:/db-derby-10.8.1.2-bin/lib"/>
</library>
```

For Informix JCC

```
<dataSource id="informixjcc" jndiName="jdbc/informixjcc">
    <jdbcDriver libraryRef="DB2JCC4Lib"/>
    <properties.informix.jcc databaseName="SAMPLEDB" serverName="localhost" portNumber="1526"/>
</dataSource>
```

```
        <library id="DB2JCC4Lib">
            <fileset dir="C:/Drivers/jcc/4.8" includes="db2jcc4.jar db2jcc_license_cisuz.jar"/>
        </library>
```

For Informix JDBC

```
        <dataSource id="informix" jndiName="jdbc/informix">
            <jdbcDriver libraryRef="InformixLib"/>
            <properties.informix databaseName="SAMPLEDB" ifxIFXHOST="localhost"
                            serverName="ol_machinename" portNumber="1526"/>
        </dataSource>

        <library id="InformixLib">
            <fileset dir="C:/Drivers/informix" includes="ifxjdbc.jar ifxjdbcx.jar"/>
        </library>
```

For Microsoft SQL Server (Microsoft JDBC driver)

```
        <dataSource id="mssqlserver" jndiName="jdbc/mssqlserver">
            <jdbcDriver libraryRef="MSJDBCLib"/>
            <properties.microsoft.sqlserver databaseName="SAMPLEDB"
                                        serverName="localhost" portNumber="1433"/>
        </dataSource>

        <library id="MSJDBCLib">
            <fileset dir="C:/sqljdbc_4.0/enu" includes="sqljdbc4.jar"/>
        </library>
```

For Microsoft SQL Server (DataDirect Connect for JDBC driver)

```
        <dataSource id="ddsqlserver" jndiName="jdbc/ddsqlserver">
            <jdbcDriver libraryRef="DataDirectLib"/>
            <properties.datadirect.sqlserver databaseName="SAMPLEDB"
                                        serverName="localhost" portNumber="1433"/>
        </dataSource>

        <library id="DataDirectLib">
            <fileset dir="C:/DataDirect/Connect-4.2/lib" includes="sqlserver.jar"/>
        </library>
```

For MySQL

```
        <dataSource id="mySQL" jndiName="jdbc/mySQL">
            <jdbcDriver libraryRef="MySQLLib"/>
            <properties databaseName="SAMPLEDB" serverName="localhost" portNumber="3306"/>
        </dataSource>

        <library id="MySQLLib">
            <fileset dir="C:/mysql-connector-java-x.x.xx"
                    includes="mysql-connector-java-x.x.xx.jar"/>
        </library>
```

For Oracle

```
        <dataSource id="oracle" jndiName="jdbc/oracle">
            <jdbcDriver libraryRef="OracleLib"/>
            <properties.oracle URL="jdbc:oracle:thin:@//localhost:1521/SAMPLEDB"/>
        </dataSource>

        <library id="OracleLib">
            <fileset dir="C:/Oracle/lib" includes="ojdbc6.jar"/>
        </library>
```

For Sybase

```
        <dataSource id="sybase" jndiName="jdbc/sybase">
            <jdbcDriver libraryRef="SybaseLib"/>
            <properties.sybase databaseName="SAMPLEDB" serverName="localhost" portNumber="5000"/>
        </dataSource>

        <library id="SybaseLib">
            <fileset dir="C:/Drivers/sybase" includes="jconn4.jar"/>
        </library>
```

For solidDB®
```
<dataSource id="solidDB" jndiName="jdbc/solidDB">
    <jdbcDriver libraryRef="solidLib"/>
    <properties databaseName="SAMPLEDB" URL="jdbc:solid://localhost:2315/"/>
</dataSource>

<library id="solidLib">
    <fileset dir="C:/Drivers/solidDB" includes="SolidDriver2.0.jar"/>
</library>
```

For a JDBC driver that is not known to the Liberty profile
```
<dataSource id="sample" jndiName="jdbc/sample" type="javax.sql.XADataSource">
    <jdbcDriver libraryRef="SampleJDBCLib"
            javax.sql.XADataSource="com.ibm.sample.SampleXADataSource"/>
    <properties databaseName="SAMPLEDB" hostName="localhost" port="12345"/>
</dataSource>

<library id=SampleJDBCLib">
    <fileset dir="C:/Drivers/SampleJDBC/" includes="sampleDriver.jar"/>
</library>
```

In the example, the JDBC driver is located at `C:/Drivers/SampleJDBC/sampleDriver.jar` and provides an implementation of `javax.sql.XADataSource` named `com.ibm.sample.SampleXADataSource`. The JDBC driver also provides vendor-specific data source properties such as **databaseName**, **hostName** and **port**.

Liberty profile: How data source configuration updates are applied

If you change the attributes of the **dataSource** element while a server is running, the updates to different attributes are applied at different times and in different ways.

You configure a data source by specifying the attributes of the **dataSource** element in the `server.xml` configuration file. If you change these attributes for a running server, the updates are applied at different times and in different ways, depending on which attribute is changed. The following table describes, for each attribute of the **dataSource** element, how a configuration change is applied at run time.

Table 4. How data source configuration updates are applied at run time. The first column of the table lists the attributes of the **dataSource** *element. The second column describes, for each attribute, how the configuration update is applied at run time.*

Attribute name	How the configuration update is applied
beginTranForResultSetScrollingAPIs	The update is effective immediately.
beginTranForVendorAPIs	The update is effective immediately.
commitOrRollbackOnCleanup	The update is effective immediately.
connectionManagerRef	All connections and the connection pool are destroyed. The data source is then managed by the new connection manager.
connectionSharing	The update is applied with each first connection handle in a transaction.
isolationLevel	The update is applied with new connection requests; current connections retain their isolation level.
jdbcDriverRef	All connections and the connection pool are destroyed. The new JDBC driver is then used.
jndiName	All connections and the connection pool are destroyed. The new JNDI name is then used.
propertiesRef	If the data source is Derby Embedded, all connections and the connection pool are destroyed before new properties go into effect. For other JDBC drivers, the new properties go into effect with new connection requests.
queryTimeout	The update is effective immediately.
recoveryAuthDataRef	Authentication data for transaction recovery.
statementCacheSize	The statement cache is resized upon next use.

*Table 4. How data source configuration updates are applied at run time (continued). The first column of the table
lists the attributes of the* `dataSource` *element. The second column describes, for each attribute, how the configuration
update is applied at run time.*

Attribute name	How the configuration update is applied
`supplementalJDBCTrace`	All connections and the connection pool are destroyed. The new setting is then used.
`syncQueryTimeoutWithTransactionTimeout`	The update is effective immediately.
`transactional`	The update is applied to new connections and existing connections not in use from the connection pool.
`type`	All connections and the connection pool are destroyed. The new setting is then used.

Adding a data source using developer tools

You can add a data source to your application using the developer tools.

Procedure

1. In the Project Explorer view, expand *liberty_profile_name* > **servers** > *server_name* > **server.xml**, and select **Open With** > **XML Editor**.
2. In the design view, select **Server Configuration**, and click **Add**.
3. On the Select item to add to Server Configuration: page, select **Data Source**.
4. On the Data Source Details panel, beside **JDBC Driver**, click **Create**.
5. On the Configure a JDBC driver page,
   - In the **Id** field, type an ID for your JDBC driver, for example `derbyEmbedded`.
   - In the **Library ref:** field, select an option from the drop-down list, or click **Create** to create a new one.
     - On the Configure a shared library page, in the **Id** field, type the ID for your JDBC driver that you created, for example `derbyEmbedded`.
     - Iin the **Fileset ref:** field, select an option from the drop-down list, or click **Create** to create a new one.
       - On the Configure a fileset page, in the **Id** field, type an ID for your JDBC driver, for example `derbyEmbedded`.
       - In the **Base directory** field, type a path for your base directory or click **Browse** to locate a library location path.
       - In the **Includes** field, type the name of your data source, for example `derby.jar`, or click **Browse** to locate the data source archive file that you want to include.
       - In the **Excludes** field, type the archives that you want to exclude, or click **Browse** to locate the data source archive file that you want to exclude. Click **Okay**
     - Click **Okay**
   - Click **Okay**
6. Your JDBC data source has been added to your application.

Liberty profile: Application-defined data sources

You can define a data source within your application, through annotations or in the deployment descriptor, as defined by the Java EE specification.

This capability is limited to names in `java:comp`. Other name spaces such as `java:module`, `java:app`, and `java:global` are not available.

When defining a data source in an application, the JDBC driver must be made available to the application. This is accomplished by configuring a shared library in the `server.xml` for your application.

For example:

```
<application id="myApp" name="myApp" location="myApp.war" type="war">
    <classloader commonLibraryRef="DB2Lib"/>
  </application>

  <library id="DB2Lib">
    <fileset dir="C:/DB2/java" includes="db2jcc4.jar db2jcc_license_cisuz.jar"/>
  </library>
```

Then you can define a data source in your application either via annotations or in the deployment descriptor.

- Use annotations as in the following example:

```
@DataSourceDefinition(
        name          = "java:comp/env/jdbc/db2",
        className     = "com.ibm.db2.jcc.DB2DataSource",
        databaseName  = "SAMPLEDB",
        serverName    = "localhost",
        portNumber    = 50000,
        properties    = { "driverType=4" },
        user          = "user1",
        password      = "pwd1"
        )

public class MyServlet extends HttpServlet {

    @Resource(lookup="java:comp/env/jdbc/db2")
    DataSource ds;
```

- Use the deployment descriptor as in the following example, for example, in a `web.xml` file:

```
<data-source>
    <name>java:comp/env/jdbc/db2</name>
    <class-name>com.ibm.db2.jcc.DB2DataSource</class-name>
    <server-name>localhost</server-name>
    <port-number>50000</port-number>
    <database-name>SAMPLEDB</database-name>
    <user>user1</user>
    <password>pwd1</password>
    <property><name>driverType</name><value>4</value></property>
  </data-source>
```

In general, properties that can be defined on `dataSource` or `connectionManager` in the `server.xml` files can also be specified on application defined data sources. However, you cannot specify properties that refer to other elements, such as `connectionManagerRef` and `jdbcDriverRef`, because the application defined data source implicitly defines the connection manager and JDBC driver. When using application-defined data sources for two-phase commit, you can specify the `recoveryAuthDataRef` property to select the authentication data that is used for transaction recovery. However, it is important to be aware that recovery of transactions is only possible while the application is running. You can use variables, encoded passwords, and duration syntax in application defined data sources.

Note: The duration syntax does not apply to properties that are explicitly defined in the annotation, such as **`loginTimeout`** or **`maxIdleTime`**.

Here is an example of two data sources using connection manager properties, variables, encoded passwords, and duration syntax.

```
@DataSourceDefinitions(value = {
    @DataSourceDefinition(
        name          = "java:comp/env/jdbc/derby",
        className     = "org.apache.derby.jdbc.EmbeddedDataSource40",
        databaseName  = "${shared.resource.dir}/data/SAMPLEDB",
        minPoolSize   = 1,
```

```
        maxPoolSize  = 10,
        maxIdleTime = 180,
        properties = { "agedTimeout=10m", "connectionTimeout=30s", "createDatabase=create" }
        ),
    @DataSourceDefinition(
        name          = "java:comp/env/jdbc/oracle",
        className      = "oracle.jdbc.pool.OracleDataSource",
        url            = "jdbc:oracle:thin:@//localhost:1521/SAMPLEDB",
        user          = "user1",
        password      = "{xor}Oz0vKDtt"
        )
})
```

Configuring connection pooling for database connections

You can configure connection pooling for your data source by defining a connection manager for it.

Example

The following example code uses the `connectionManager` element in the `server.xml` file to define a connection pool for a data source:

```
<dataSource id="ds1" jndiName="jdbc/example" jdbcDriverRef="DB2" >
  <connectionManager maxPoolSize="10" minPoolSize="2" />
</dataSource>
```

The server uses default values for any connection management settings that are not defined on the connection manager element. If a connection manager is not defined at all for a data source, the server uses default values for all of the settings.

You can define multiple data sources and associate each with a different connection manager. However, you cannot associate multiple data sources with a single connection manager.

For more information about the `connectionManager` element, see "Liberty profile: Configuration elements in the server.xml file" on page 3.

Liberty profile: How connection pooling configuration updates are applied:

If you change the attributes of the **connectionManager** element while a server is running, the updates to different attributes are applied at different times and in different ways.

You configure a connection pool by specifying the attributes of the **connectionManager** element in the `server.xml` configuration file. If you change these attributes for a running server, the updates are applied at different times and in different ways, depending on which attribute is changed. The following table describes, for each attribute of the **connectionManager** element, how a configuration change is applied at run time.

Table 5. How connection manager configuration updates are applied at run time. The first column of the table lists the attributes of the **connectionManager** *element. The second column describes, for each attribute, how the configuration update is applied at run time.*

Attribute name	How the configuration update is applied
`agedTimeout`	The update is effective immediately.
`connectionTimeout`	The update is effective immediately.
`maxIdleTime`	The update is effective immediately.
`maxNumberOfMCsAllowableInThread`	The update is effective immediately.
`maxPoolSize`	The update is effective immediately.
`minPoolSize`	The update is effective immediately.
`numConnectionsPerThreadLocal`	The update is effective immediately.

Table 5. How connection manager configuration updates are applied at run time (continued). The first column of the table lists the attributes of the **connectionManager** *element. The second column describes, for each attribute, how the configuration update is applied at run time.*

Attribute name	How the configuration update is applied
reapTime	The update is effective immediately.
purgePolicy	The update is effective immediately.

Note: The attributes **agedTimeout** and **maxIdleTime** are updated immediately. However, they are not used fully unless the value of **reapTime** attribute is greater than zero.

Because updates to the connection manager are effective immediately, errors might occur if you make changes with active connections; including the potential risks for the connections to be ended prematurely.

Liberty profile: How database transactions are recovered

Support for transactions is provided by the transaction service. Recovery can occur either when the transaction service is first used, or at server startup. When recovering any in-doubt database transactions, the Liberty profile transaction manager uses either the unique identifier or the JNDI name to locate the current `dataSource` element.

By default, transaction recovery after a server failure happens when the transaction service is first used (at first lookup of a UserTransaction), rather than at server startup. This behaviour can be altered by using a pair of configuration attributes for the transaction service. These attributes control when recovery happens, and whether the system waits for recovery to finish before allowing transactional work to proceed. You set these attributes in the `server.xml` file. For example, the following settings specify that recovery should occur at server startup, and that the server should wait for recovery to finish before allowing transactional work to proceed:

```
<transaction
  recoverOnStartup="true"
  waitForRecovery="true"
/>
```

You configure a data source by specifying the attributes of the `dataSource` element in the `server.xml` configuration file. You also assign a unique identifier or a **jndiName** attribute for the data source as follows:

```
<dataSource id="ds1" jndiName="jdbc/ds1"... />
```

You must not change the value of the **id** or **jndiName** attribute when a recovery is pending for a transaction in which the data source participated. If you make changes to any other attributes of the `dataSource` element, those changes are honored for the recovery. This enables you to, for example, add a **recoveryAuthDataRef** attribute that specifies a database user ID and password to use for recovery.

The database user ID and password to use for recovery are determined according to the following precedence:

1. If the `dataSource` element has the **recoveryAuthDataRef** attribute defined, then the user ID and password from the `authData` element are used. For example:

   ```
   <authData id="recoveryAuth" user="dbuser1" password="{xor}Oz0vKDtu"/>
   <dataSource id="ds1" jndiName="jdbc/ds1" jdbcDriverRef="DB2"
               recoveryAuthDataRef="recoveryAuth" .../>
   ```

2. Otherwise, if container managed authentication is used, then the user ID and password from the container managed authentication alias are used. For example:

   - In the `ibm-web-bnd.xml` file, you have the following code:

     ```
     <resource-ref name="jdbc/ds1ref" binding-name="jdbc/ds1">
         <authentication-alias name="user1Auth"/>
      </resource-ref>
     ```

   - In the `server.xml` file, you have to define the following code:

```
        <authData id="user1Auth" user="dbuser1" password="{xor}Oz0vKDtu"/>
        <dataSource id="ds1" jndiName="jdbc/ds1" jdbcDriverRef="DB2" .../>
```

3. Otherwise, the user ID and password from the `dataSource` element are used. For example:

```
<dataSource id="ds1" jndiName="jdbc/ds1" jdbcDriverRef="DB2" ...>
     <properties.db2.jcc databaseName="testdb" user="dbuser1" password="{xor}Oz0vKDtu"/>
</dataSource>
```

4. If none of the above are specified, and the recovery is attempted without any user ID and password, then the behavior is determined by the JDBC driver and database.

Note: If the transaction recovery is performed by an application-defined data source, such as an `@DataSourceDefinition` annotation or a `<data-source>` element in the deployment descriptor, you have to make sure the application is running for the data source when the recovery is happening. Configurations in the `server.xml` file cannot be used to recover application-defined data sources.

Administering web applications on the Liberty profile

The Liberty profile provides support to the web applications using server features such as `servlet-3.0`, `jsp-2.2` and other features.

Procedure

Specify when Servlets are loaded and initialized.

Specifying when servlets are loaded and initialized

By default, the Liberty profile defers servlet loading until a request is received for the associated web application. You can override this default behavior by specifying the web container **deferServletLoad** attribute to `false`.

About this task

The servlet specification defines the **load-on-startup** servlet attribute, which is specified in the `web.xml` file of a web application. If a servlet has a non-negative value for the **load-on-startup** attribute, the servlet must be loaded and initialized when the web application is deployed. The Liberty profile optimizes server start time and memory use by not starting a servlet until a request is received for the web application. You can override this deferral so that your servlets are loaded and initialized when the web application is installed, rather that waiting for the first request for the application.

Example

To configure the server to load servlets when a web application is installed, add the following line to the `server.xml` configuration file or a file that it includes:

```
<webContainer deferServletLoad="false"/>
```

This setting applies to all web applications installed in the server.

Chapter 3. How do I administer applications and their environments?

Follow these shortcuts to get started quickly with popular tasks.

When you visit a task in the information center, look for the **IBM Suggests** feature at the bottom of the page. Use it to find available tutorials, demonstrations, presentations, developerWorks® articles, IBM Redbooks®, support documents, and more.

Create WebSphere profiles

Register a node with the administrative agent

Administer nodes using the administrative agent

Administer configurations

Administer application servers with the console

Configure application servers with scripting

Manage application servers with scripting

Administer generic servers

Administer custom services

Administer the UDDI registry

Use the console to administer communication with web servers (plug-ins)

Use scripting to administer communication with web servers (plug-ins)

Administer HTTP sessions with the console

Administer HTTP sessions with scripting

Provide access to naming and directory resources (JNDI) - Name server

Provide access to naming and directory resources (JNDI) - Bindings

Provide access to relational databases (JDBC resources) with the console

Provide access to relational databases (JDBC resources) with scripting

Choose a messaging provider.

Provide access to messaging resources (default messaging provider) with scripting

Install applications with the console

Install applications with scripting

Start and stop applications with the console

Start and stop applications with scripting

Update applications with the console

Update applications with scripting

Deploy and administer web services applications

Administer business-level applications using the administrative console

Administer business-level applications using programming

Set up business-level applications using scripting

Manage environment configurations with properties files using wsadmin scripting

Choose an administrative client

Use the administrative console

Using scripting (wsadmin)

Troubleshoot deployment

Troubleshoot administration

Chapter 4. Using the administrative clients

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.

Procedure

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.

Administrative console

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 produce 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 generated 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.
- **Restart**. Stops the selected objects and starts them again.
- **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.

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.

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

Before reading this section, read the terms and information in Console layout.

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.

Table 6. Console navigation tree control functions. 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

Table 7. Console help control functions.   The following table describes the console help control functions.

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.
⬆ ⬇	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 Versions 3.0.x and 4.0.x
* Firefox Version 1.5, for AIX V6.1 only
* Microsoft Internet Explorer Versions 8.0 and 9.0

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.

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
  - **Windows** Supports interfaces commonly used by screen readers and screen magnifiers
  - 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.

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.

Information	Value
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.

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`/profiles/`profile_name`/config/cell/`cell_name`/applications/isclite.ear/ deployments/isclite/isclite.war/WEB-INF. 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 8. Console identity key values.   The following table describes the 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

Table 8. Console identity key values (continued). The following table describes the console identity key values.

Key (id)	Initial setting (value)	Description
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	*empty string*	A custom string to be displayed in the banner and title bar.
ISC.CONSOLE.ID.DEPLOYER.*unique_suffix*	*None*	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.*unique_suffix* id attribute, you can provide multiple identity strings for console users to select from. Each entry of this type should have a distinct *unique_suffix* from the other entries. If multiple entries are provided with the same *unique_suffix*, then only the first value with this suffix is displayed in the selection list for the console user.

Examples:

– 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.

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.

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.

| Linux | HP-UX | Solaris | AIX | The deployConsole.py script is located in the *profile_root*/`bin` directory.

| Windows | The deployConsole.py script is located in the *profile_root*\`bin` directory.

Procedure

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

  | Windows |
  ```
  wsadmin.bat -f deployConsole.py install
  ```
  | Linux | HP-UX | Solaris | AIX |
  ```
  wsadmin.sh -f deployConsole.py install
  ```

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

  | Windows |
  ```
  wsadmin.bat -f deployConsole.py remove
  ```
  | Linux | HP-UX | Solaris | AIX |
  ```
  wsadmin.sh -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

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

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.

Procedure

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, change the port number in the *installation root*/profiles/*profile name*/config/cells/*cell_name*/nodes/*node_name*/serverindex.xml file and the *installation root*/profiles/*profile name*/config/cells/*cell_name*/virtualhosts.xml files. Change all the occurrences of the console port to a new port number. 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.

      Important: `Windows` If the application server is on Windows Server 2008 and you are accessing the console with Microsoft Internet Explorer V8.0, disable the enhanced security configuration in the browser. Otherwise, the console does not work correctly. For further information about enhanced security, read material on the Microsoft website.

   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 stand-alone 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.

Save changes to the master repository

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.

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:

Procedure

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.

An example follows with workspace automatic refresh turned on and two users logged in to the console at the same time. If the first user deletes a mail provider from the collection without saving his changes, the second user does not see the changes. The first user must save the changes. Then the second user must refresh the collection panel on his own console to see the changes. However, if the automatic refresh is not on, then the second user will not see the deletion of the mail provider on the collection panel until after he logs out and logs back in.

Information	Value
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.

Information	Value
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.

Information	Value
Default	false (cleared)

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

Information	Value
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.

Information	Value
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.

Information	Value
Default	false (cleared)

Bidirectional support options: Specifies bidirectional (Bidi) text preferences for the administrative console.

Information	Value
Default	false (cleared)

Bidirectional support options
Use the Bidirectional support options page to specify bidirectional (Bidi) text preferences for the administrative console.

Bidirectional support means that text is supported going in both directions for different types of alphabets. WebSphere Application Server presents Bidi text using left-to-right (LTR) orientation in most languages. However, when Bidi text is used as part of mixed Bidi and Latin text, for example, the preferred orientation is right-to-left (RTL).

For text entry fields, this means that the cursor is placed at the right side and moves to the left as characters are typed in an RTL alphabet, such as Hebrew or Arabic.

Bidi support helps to maintain visual structure in complex fields, such as file paths, emails, URLs and Xpaths.

If you select **Enable bidirectional support for all users** under Global Preferences, any change made affects all users. Note that if you select this option that **Enable bidirectional support for this user** under Current® User Preferences is automatically selected also. However, if you only select **Enable bidirectional support for this user**, any changes made only affect the user that is currently logged in.

To view this administrative console page, click **Environment > Console preferences > Bidirectional support options**.

Enable bidirectional support for all users:

Select this choice to enable bidirectional support for all users.

Information	Value
Default	false

Default text direction for all users:

Select one of the options on this menu to indicate which direction the text should go for all users.

The default text direction is left to right. This is the natural base text direction for most languages, including European, Asian and Indic languages

If you select right to left, this is the natural base text direction for languages such as Arabic, Hebrew, Urdu and Farsi.

If you select contextual, the base text direction is set according to the first strong character. This is an appropriate choice when at the time of setting the base direction, the main language of the string is not yet known.

Information	Value
Default	LTR

Enable bidirectional support for this user:

Select one of the options in this menu to indicate which direction the text should go for only the user currently logged in.

Information	Value
Default	false

Text direction for this user: The default text direction is left to right. This is the natural base text direction for most languages, including European, Asian and Indic languages

If you select right to left, this is the natural base text direction for languages such as Arabic, Hebrew, Urdu and Farsi.

If you select contextual, the base text direction is set according to the first strong character. This is an appropriate choice when at the time of setting the base direction, the main language of the string is not yet known.

Information	Value
Default	LTR

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.

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.

Display inherit policy set attachments confirmation: Select the check box if you want to enable the inherit policy set attachments confirmation.

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 (base) and WebSphere Application Server, Express, a resource can be visible in the administrative console collection table at the cell, node, or server scope.

For WebSphere Application Server (base) and WebSphere Application Server, 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 (base) and WebSphere Application Server, 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 (base) and WebSphere Application Server, 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 (base) and WebSphere Application Server, Express, you can configure resources and the product variables under all four scopes.

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 helps 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.

Procedure

- 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.

      Attention: 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 website.

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.

Procedure

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 Javascript. If your browser supports Javascript, the Command assistance window automatically refreshes the command list to reflect the most recent console action. If the browser does not support Javascript, 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.

Table 9. Console component actions. The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Applications	- List the applications. - Install the application. - Update the complete application. - Start the application. - Stop the application. - Edit the deployment target mapping. - List Structured Query Language in Java (SQLJ) profiles. - Customize and bind SQLJ profiles. - List IBM Optim™ pureQuery Runtime bind files (*). - Bind IBM Optim pureQuery Runtime bind files (*). - Uninstall the application. (*) Command assistance in the administrative console produces wsadmin commands for SQLJ. These commands work with IBM Optim pureQuery Runtime bind files. Command assistance does not produce separate commands for the IBM Optim pureQuery Runtime bind files. However, in the wsadmin environment separate commands exists for you to use when working with IBM Optim pureQuery Runtime bind files.
Web servers	- 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.
Channel framework	- 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action

Servers	• 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.

Servers (continued)	• 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.

Table 9. Console component actions (continued).   The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Servers (continued)	• 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.
Servers (continued)	• 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.

Table 9. Console component actions (continued).   The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action

Servers (continued)	• 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	• 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.

Table 9. Console component actions (continued).   The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Environment	• 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)	• 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.

Table 9. Console component actions (continued).   The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Resources	- 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.
Resources (continued)	- 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Resources (continued)	• 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)	• 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.

Table 9. Console component actions (continued).   The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Transaction services	• 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	• 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.

Table 9. Console component actions (continued).   The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Security (continued)	- 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Security (continued)	• 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Security (continued)	• 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.

Table 9. Console component actions (continued).   The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Security (continued)	• 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Security (continued)	• 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Security (continued)	• 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. • Configure programmatic session cookies.

Table 9. Console component actions (continued).   The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Service integration	• 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Service integration (continued)	• 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Service integration (continued)	• 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	• 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	• 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Web services policy sets	• 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	**Attention:** Before you use the generated wsadmin command for policies, see the documentation for the PolicySetManagement command group for the AdminTask object. • 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	• 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.
Web services trust service	• 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.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Web Services Security distributed cache	• 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	• 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	• 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	• Re-initialize the Runtime Performance Advisor tool. • Set the Runtime Performance Advisor tool attributes.
Portlets and portlet containers	• 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.
Replication Domains	• List data replication domain members.

Table 9. Console component actions (continued). The following table lists the components and the actions in the administrative console that have command assistance for a particular component.

Component	Action
Users and groups	- 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.

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.

Procedure

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 JACL 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.

Procedure

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
```

Getting started with wsadmin scripting

Scripting is a non-graphical alternative that you can use to configure and manage WebSphere Application Server.

About this task

The WebSphere Application Server wsadmin tool provides the ability to run scripts. The wsadmin tool supports a full range of product administrative activities.

Restriction: The wsadmin tool does not apply to the Liberty profile. See Administering the Liberty profile from the command prompt instead.

The following figure illustrates the major components involved in a wsadmin scripting solution:
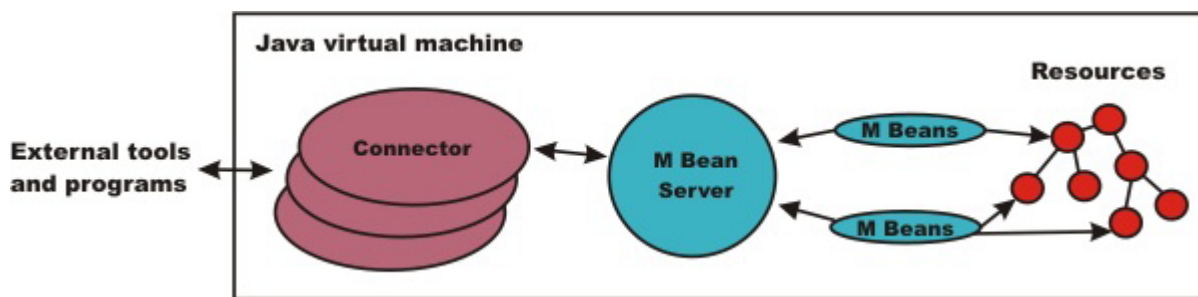
Figure 1. A WebSphere Application Server scripting solution. This figure illustrates the major components involved in a wsadmin scripting solution.

The wsadmin tool supports two scripting languages: Jacl and Jython. Five objects are available when you use scripts:

- **AdminControl**: Use to run operational commands.
- **AdminConfig**: Use to run configurational commands to create or modify WebSphere Application Server configurational elements.
- **AdminApp**: Use to administer applications.
- **AdminTask**: Use to run administrative commands.
- **Help**: Use to obtain general help.

The scripts use these objects to communicate with MBeans that run in WebSphere Application Server processes. MBeans are Java objects that represent Java Management Extensions (JMX) resources. JMX is an optional package addition to Java 2 Platform Standard Edition (J2SE). JMX is a technology that provides a simple and standard way to manage Java objects.

Important: Some wsadmin scripts, including the AdminApp install, AdminApp update, and some AdminTask commands, require that the user ID under which the server is running must have read permission to the files that are created by the user that is running wsadmin scripting. For example, if the application server is running under user1, but you are running wsadmin scripting under user2, you might encounter exceptions involving a temporary directory. When user2 runs wsadmin scripting to deploy an application, a temporary directory for the enterprise application archive (EAR) file is created. However, when the application server attempts to read and unzip the EAR file as user1, the process fails. It is not recommended that you set the umask value of the user that is running wsadmin scripting to 022 or 023 to work around this issue. This approach makes all of the files that are created by the user readable by other users. To resolve this issue, consider the following approaches based on your administrative policies:

- Run wsadmin scripting with the same user ID as the user that runs the deployment manager or application server. A root user can switch the user ID to complete these actions.
- Set the group ID of the user that is running the deployment manager or application server to be the same group ID as the user that is running wsadmin scripting. Also, set the umask value of the user that is running the wsadmin scripting to be at least a umask 027 value so that files that are created by the wsadmin scripting can be read by members of the group.
- Run wsadmin scripting from a different machine. This approach forces files to be transferred and bypasses the file copy permission issue.

To perform a task using scripting, you must first perform the following steps:

Procedure

1. Choose a scripting language. The wsadmin tool only supports Jacl and Jython scripting languages. Jacl is the language specified by default. If you want to use the Jython scripting language, use the -lang option or specify it in the wsadmin.properties file.
2. Start the wsadmin scripting client interactively, as an individual command, in a script, or in a profile.

What to do next

Before you perform any task using scripting, make sure that you are familiar with the following concepts:
- Java Management Extensions (JMX)
- WebSphere Application Server configuration model
- wsadmin tool
- Jacl syntax or Jython syntax
- Scripting objects

Optionally, you can customize your scripting environment. For more information, see Administrative properties for using wsadmin scripting.

After you become familiar with the scripting concepts, choose a scripting language, and start the scripting client, you are ready to perform tasks using scripting.

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.

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

Improved administrative scripting features

Command name	Description

Overview and new features for scripting the application serving environment

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

"What is new for scripted administration (wsadmin)"

> This topic provides an overview of new and changed features for administrative scripting and the wsadmin tool.

Introduction: Administrative scripting (wsadmin)

> This topic provides an introduction to administrative scripting and the wsadmin tool.

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.

Procedure

- 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 (JMX) for WebSphere Application Server

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 lowest 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 highest 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 highest 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).
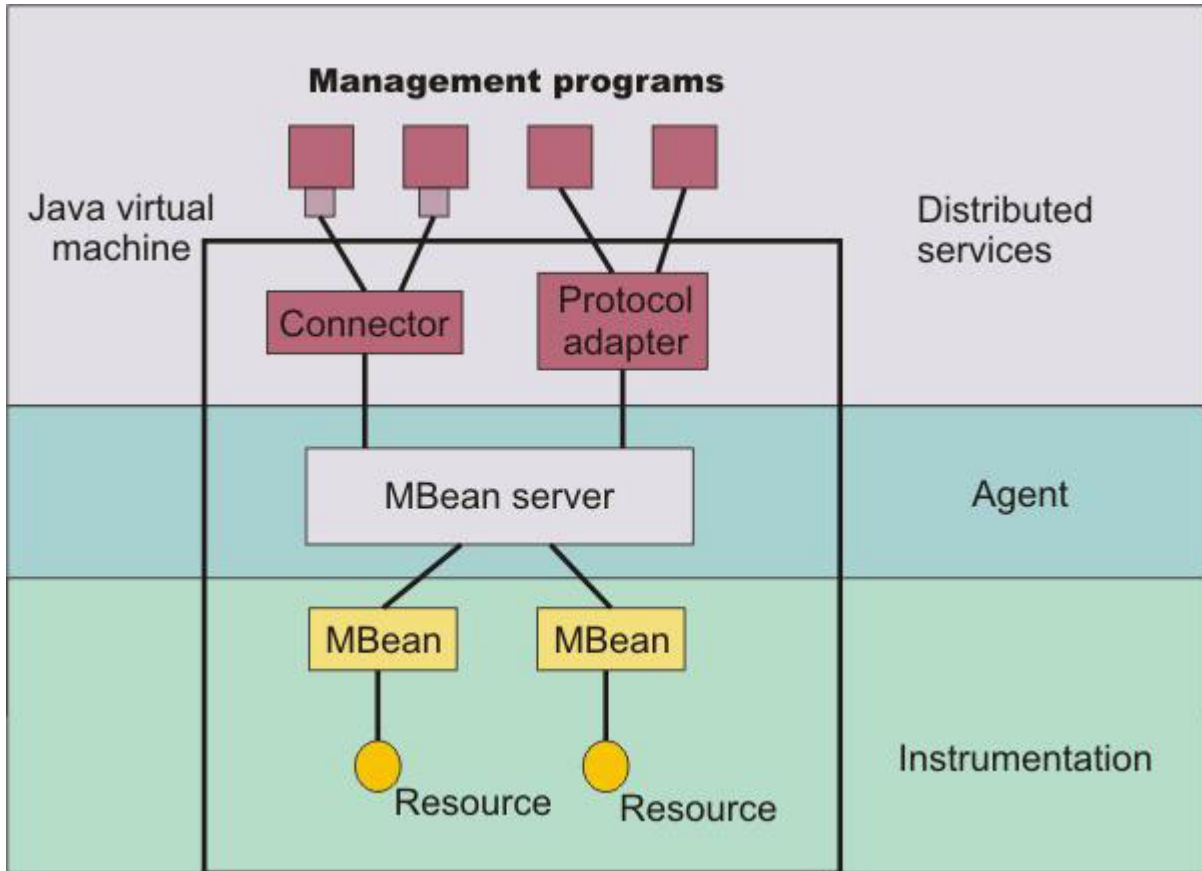
Java Management Extensions in WebSphere Application Server

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.
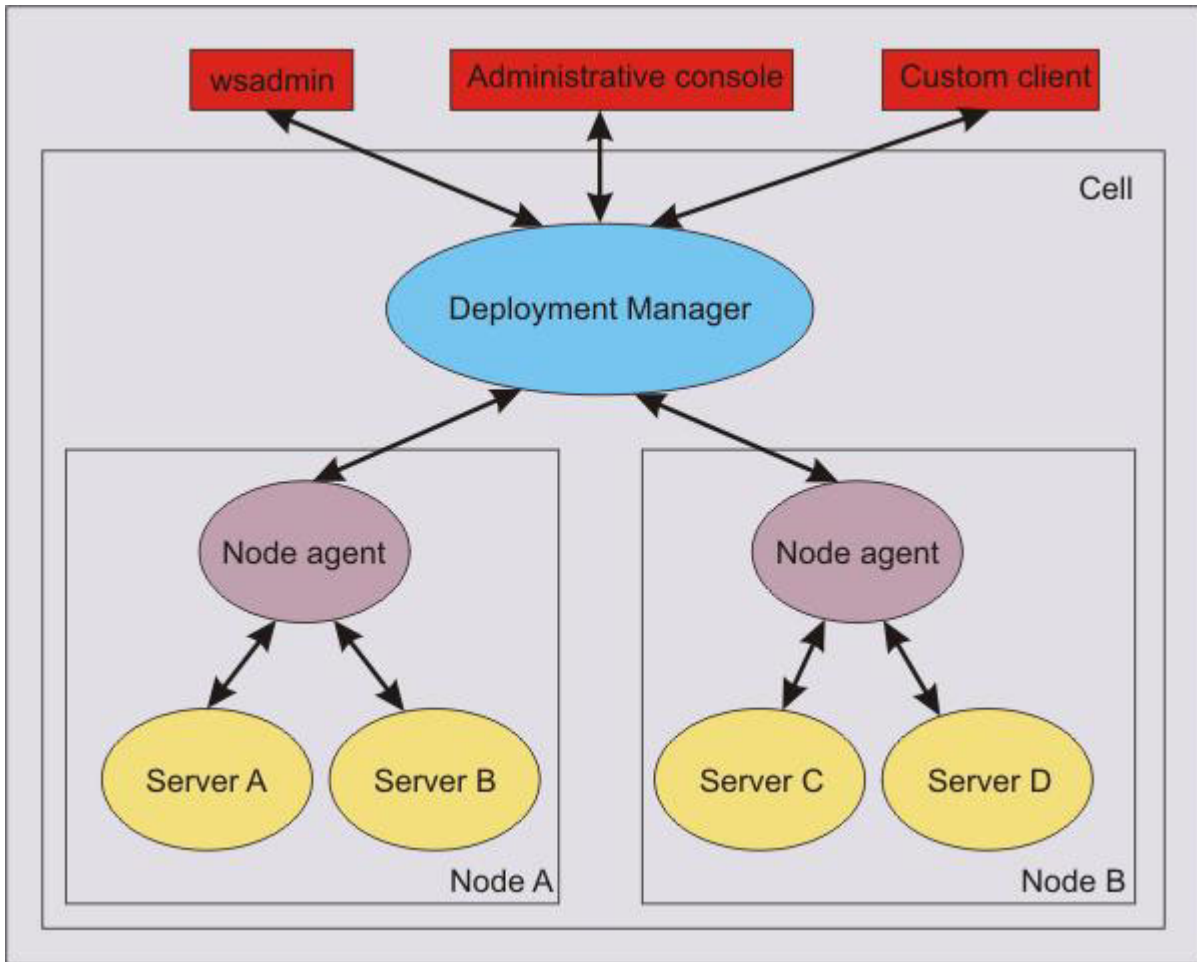
Creating a custom Java administrative client program using WebSphere Application Server administrative Java APIs

This section describes how to develop a Java program for accessing the WebSphere Application Server administrative system by using the product 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.

Procedure

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

Product 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 on MBean programming, see MBean Java API documentation.

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.

Procedure

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 for the Simple Object Access Protocol (SOAP) connector:

   ```
   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 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.

- The following example shows how to create an AdminClient instance for the Remote Method Invocation (RMI) connector. Some commands are split on multiple lines for printing purposes.

```
Properties connectProps = new Properties();
connectProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_RMI);
connectProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
connectProps.setProperty(AdminClient.CONNECTOR_PORT, "2809");
connectProps.setProperty(AdminClient.USERNAME, "test2");
connectProps.setProperty(AdminClient.PASSWORD, "user24test");
System.setProperty("com.ibm.CORBA.ConfigURL",
 "file:C:/AA/cf010839.26/profiles/AppSrv02/properties/sas.client.props");
System.setProperty("com.ibm.SSL.ConfigURL",
 "file:C:/AA/cf010839.26/profiles/AppSrv02/properties/ssl.client.props");
AdminClient adminClient = null;
try
{
        adminClient = AdminClientFactory.createAdminClient(connectProps);
}
catch (ConnectorException e)
{
        System.out.println("Exception creating admin client: " + e);
}
```

Note: When you use the createAdminClient method within application code that runs on an application server, such as within servlets and JavaServer Pages (JSP) files, you must set the CACHE_DISABLED property to true. For example:

```
connectProps.setProperty(AdminClient.CACHE_DISABLED, "true");
```

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 Remote Method Invocation connector to reach the server; for the connector type, use the value: AdminClient.CONNECTOR_TYPE_RMI.

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 RMI connector is listening.

In a single server installation, the default port number for the application server RMI connector is 2809. In a WebSphere Application Server, Network Deployment installation, the default port number for the deployment manager RMI connector is 9809.

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, you need to set a system property to point to the ssl.client.props file and the sas.client.props file. If you run on a local machine you can point to the actual location. If you run on a remote machine, you can copy these properties files from the server machine and put them anywhere you want, specifying the path to where you put the files.

You can specify a user name and password inside the sas.client.props file, When you do, specify `com.ibm.CORBA.loginSource=properties`. If you want to set the user name and password inside your client program, specify `com.ibm.CORBA.loginSource=none` in the sas.client.props file.

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:

Table 10. Key property descriptions. Key properties include types, name, cell, node, and process.

Key Property	Description
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 the product 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.

Table 11. Examples object names with wildcard key properties. Include asterisks (*) to specify wildcard key properties.

Object name	Description
: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 product 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("****************************************************");
}
```

Results

The administrative client can handle event notifications that are emitted from an MBean.

Note: If a client program registers a notification listener through an RMI or JSR160RMI connector and the ORB thread does not stop running, and thus prevents the Java virtual machine from exiting, add a `System.exit()` statement to the client program. The ORB starts a thread to handle notification propagation to the client. This thread does not automatically exit with the client main thread unless the main thread has a `System.exit()` statement. Place a `System.exit()` statement in a location in the client program that enables the ORB thread and main thread to stop processing. For example, place the `System.exit()` statement in a `catch` or `finally` clause of the client program main `try` block.

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

```java
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 ntfyCount = 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:

The following examples are complete administrative client program. The second examples uses a Remote Method Invocation (RMI) connector.

Both of these examples show 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 AdminClientExampleSoap implements NotificationListener
{

    private AdminClient adminClient;
    private ObjectName nodeAgent;
    private long ntfyCount = 0;

    public static void main(String[] args)
    {
        AdminClientExampleSoap ace = new AdminClientExampleSoap();

        // Create an AdminClient
        ace.createAdminClient();

        // Find a NodeAgent MBean
        ace.getNodeAgentMBean("timmieNode07");

        // 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, "timmie");
        connectProps.setProperty(AdminClient.CONNECTOR_PORT, "8886");
        System.setProperty("com.ibm.SSL.ConfigURL", "file:C:/MyThinClient/70properties/ssl.client.props");
        System.setProperty("com.ibm.SOAP.ConfigURL", "file:C:/MyThinClient/70properties/soap.client.props");

        // 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);
        }catch (Exception e){
         e.printStackTrace();
            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)
            {
            }
    }

}
```

The following example uses a RMI connector. The connector type and port value are different from what appears in the preceding example.

```
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 AdminClientExampleSoap implements NotificationListener
{

    private AdminClient adminClient;
    private ObjectName nodeAgent;
    private long ntfyCount = 0;

    public static void main(String[] args)
    {
        AdminClientExampleSoap ace = new AdminClientExampleSoap();

        // Create an AdminClient
        ace.createAdminClient();

        // Find a NodeAgent MBean
        ace.getNodeAgentMBean("timmieNode07");

        // 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, "timmie");
        connectProps.setProperty(AdminClient.CONNECTOR_PORT, "8886");
        System.setProperty("com.ibm.SSL.ConfigURL", "file:C:/MyThinClient/70properties/ssl.client.props");
        System.setProperty("com.ibm.SOAP.ConfigURL", "file:C:/MyThinClient/70properties/soap.client.props");

        // 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);
        }catch (Exception e){
         e.printStackTrace();
            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.

Procedure

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 the JSR 160: Java Management Extensions (JMX) Remote API at 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. Complete this task to develop a JMX remote client program using the JMX remote API so that you can administer your environment through JMX programming.

Procedure

1. Specify the JMX connector address for the server through the JMXServiceURL class.

   The value of the JMX service URL is:

   ```
   service:jmx:rmi://" + host + ":" + port + "/jndi/JMXConnector"
   ```

   For example, if the target server host is `sales.xyz.com` and the listening port is `1234`, the JMX service URL is:

   ```
   service:jmx:rmi://sales.xyz.com:1234/jndi/JMXConnector
   ```

   You can find the value for *port* in the Ports table of the console server settings page or in the `serverindex.xml` file that includes the target server. If the URL does not specify a value for *host*, the product uses the default value of `localhost`. If the URL does not specify a value for *port*, the product uses the default value of `2809`.

   When connecting to an administrative agent, add the administrative agent JMX connector port number to the end of the URL. For example, if the administrative agent JMX connector host is `sales.xyz.com` and the port is `6789`, then use the following URL:

   ```
   service:jmx:rmi://sales.xyz.com:6789/jndi/JMXConnector6789
   ```

2. Set the Java Naming and Directory Interface (JNDI) provider URL property to use the administrative name service for the product.

The JNDI provider URL property is javax.naming.Context.PROVIDER_URL. The administrative name service is WsnAdminNameService.

3. If the client uses security, set the -Dcom.ibm.CORBA.ConfigURL and -Dcom.ibm.SSL.ConfigURL system properties in the client Java virtual machine (JVM).

Without the -Dcom.ibm.CORBA.ConfigURL and -Dcom.ibm.SSL.ConfigURL system properties set to valid system properties files, the client does not work properly when security is enabled. The recommended way to run the JMX connector client is as an administrative thin client.

- -Dcom.ibm.CORBA.ConfigURL=file:*app_client_root*/properties/sas.client.props

- -Dcom.ibm.SSL.ConfigURL=file:*app_client_root*/properties/ssl.client.props

Typically, you can copy the properties files from an installation profile directory, preferably from the target server profile directory.

4. Specify the user ID and password for the server, if security is enabled.

5. Establish the JMX connection.

6. Get the MBean server connection instance.

Example

Use the following thin client code example to create and use the JMX client.

Some statements are split on multiple lines for printing purposes.

```
import java.io.File;
import java.util.Date;
import java.util.Set;
import java.util.Hashtable;

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 ObjectName jvm;
    private long ntfyCount = 0;
    private static String userid = null;
    private static String pwd = null;

    public static void main(String[] args)
    {
        try {

            JMXRemoteClientApp client = new JMXRemoteClientApp();

            String host=args[0];
            String port=args[1];
            String nodeName =args[2];
            userid =args[3];
            pwd = args[4];

            client.connect(host,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 host,String port) throws Exception
{
    String jndiPath="/WsnAdminNameService#JMXConnector";

    JMXServiceURL url =
        new JMXServiceURL("service:jmx:iiop://"+host+"/jndi/corbaname:iiop:"+host+":"+port+jndiPath);

    Hashtable h  = new Hashtable();

    //Specify the user ID and password for the server if security is enabled on server.

    System.out.println("Userid is " + userid);
    System.out.println("Password is " + pwd);
    if ((userid.length() != 0) && (pwd.length() != 0)) {
            System.out.println("adding userid and password to credentials...");
            String[] credentials = new String[] {userid , pwd };
            h.put("jmx.remote.credentials", credentials);
    } else {
            System.out.println("No credentials provided.");
    }


    //Establish the JMX connection.

    JMXConnector jmxc = JMXConnectorFactory.connect(url, h);

    //Get the MBean server connection instance.

    mbsc = jmxc.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();
                System.out.println("NodeAgent mbean found "+ nodeAgent.toString());
        } else {
            System.out.println("Node agent MBean was not found");
            System.exit(-1);
        }
    } catch (Exception e) {
        System.out.println(e);
        System.exit(-1);
    }
}


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 product 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, see MBean Java application programming interface (API) documentation.

Procedure

1. Create custom JMX MBeans.

    You have some alternatives to select from, when creating MBeans to extend the product 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 product 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 product 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 product administrative system. An ExtensionMBean provider implies that you supply an XML file that contains an MBean Descriptor based on the DTD shipped with the product. 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, the product 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 product 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 product 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 product 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 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 topic 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.

The product provides a special runtime 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.

The MBean registration and capabilities are as follows:

Table 12. MBean registration and capabilities. Examine the registration and capabilities for an MBean type.

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 product 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.

Procedure

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 runtime 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 product administrative service.

Example

The following example shows how to create and register a standard MBean with the 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 unicall option and the snoopy method is set to use the multicall 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.

gotcha: If you are running in a multiple JVM environment you must include the type property in the MBean descriptor.

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="unicall"/>

 <operation name="snoopy" role="operation"  type="void" targetObjectType="objectReference"
   impact="ACTION" proxyInvokeType="multicall">
  <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 full path to the MBean descriptor file


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.

Procedure

- 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.
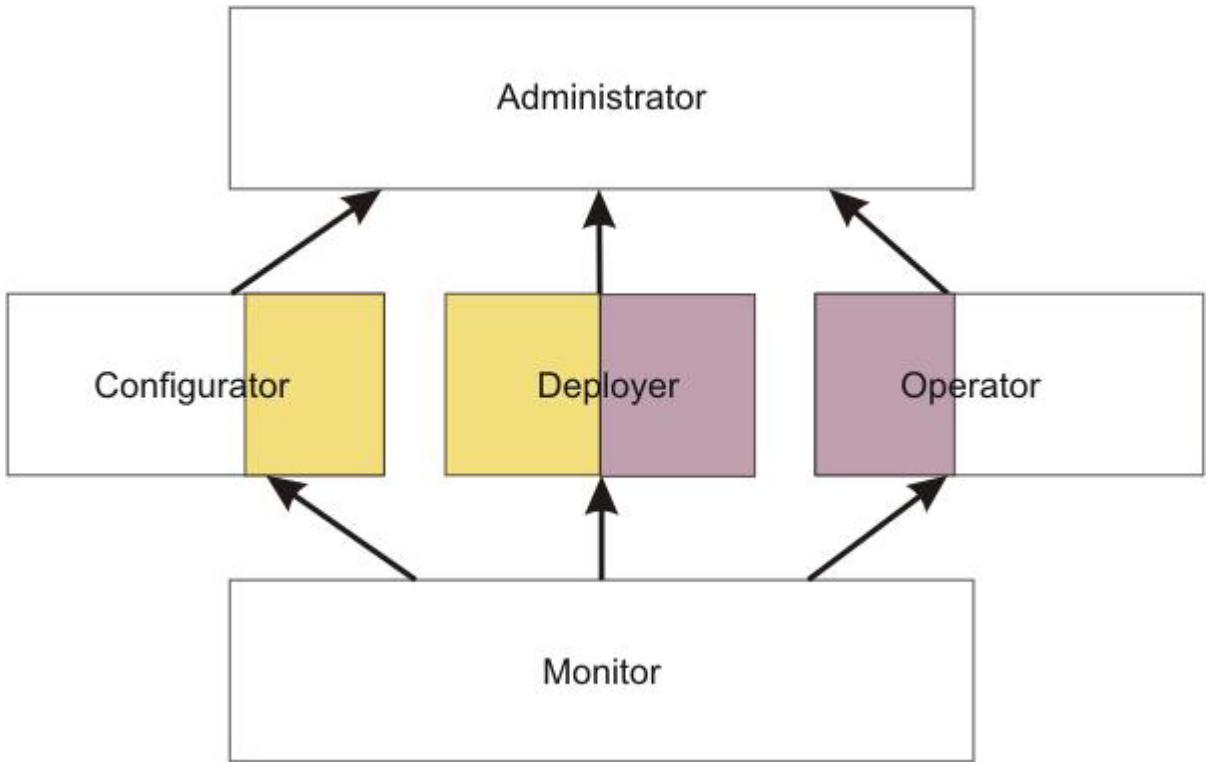
Each MBean method or operation is assigned an impact attribute with a value of either `INFO` or `ACTION`. Here are some examples:

- 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:

Table 13. Configuration MBean method impact values and security roles. An X indicates that the MBean method requires the role by default.

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:

Table 14. Operation MBean method impact values and security roles. An X indicates that the MBean method requires the role by default.

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:

Table 15. Deployer MBean method impact values and security roles. An X indicates that the MBean method requires the role by default.

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.

Procedure

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.

Procedure

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 resourceIndentifierKey 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 the product 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

best-practices: 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 product 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.

Procedure

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 (JMX) interoperability

Starting with Version 6.0, WebSphere Application Server implements Java Management Extensions (JMX) Version 1.2.

Differences between Version 6.0.x, and Version 6.1 and later

The product supports communication with earlier levels of the server through the SOAP connector. The earlier levels of the server cannot be more than two releases earlier. Version 6 and later servers can communicate with each other through an RMI connector, with one restriction. The javax.management.MBeanInfo class and its subclasses do not interoperate between Version 6.1.02 and version 6.0.2. You will receive a java.lang.IllegalArgumentException exception when attempting to send any instance of these classes between a Version 6.1 and Version 6.0.2. This restriction affects clients that perform the following operations.

- A wsadmin scripting client that tries to perform the following operations, for example in Jacl,:

```
$Help attributes MBeanObjectName
$Help operations MBeanObjectName
$Help notifications MBeanObjectName
```

- Any Java client that tries to call:

```
com.ibm.websphere.management.AdminClient.getMBeanInfo(ObjectName name);
com.ibm.websphere.management.AdminService.getMBeanInfo(ObjectName name);
javax.management.MBeanServer.getMBeanInfo(ObjectName name);
```

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.

Information	Value
Property name	jmx.serial.form
Data type	string
Value	1.2.0 or 1.2.1

To access the JVM custom properties page, go to:

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

The product 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.

Running command line tools on the Microsoft Windows Vista or later Microsoft operating system:
`Vista` `2008` `Windows 7` On the Windows Vista, Windows Server 2008, and Windows 7 operating systems, you can install WebSphere Application Server as either Administrator or non-Administrator. When it is installed as Administrator, certain operations (such as those involving Windows Services) require Administrator privileges. To ensure that WebSphere Application Server command-line tools have sufficient privileges, run them with elevated administrator authority on systems that have the Windows User Account Control (UAC) **Run all administrators in Admin Approval Mode** policy enabled. When you run these command-line tools from a Command Prompt, run them from a Command Prompt window that is launched by performing the following actions:

- Right-click a Command Prompt shortcut.
- Click **Run As Administrator**.

  When you open the Command Prompt window as Administrator, an operating-system dialog appears that asks you if you want to continue. Click **Continue** to proceed.

Examples of these tools are the `startServer` command, `stopServer` command, `firststeps` command, and the Profile Management Tool. If you are using a Windows Server Core installation of Windows Server 2008, any WebSphere Application Server commands that require a graphical interface are not supported since a Windows Server Core system does not have a graphical user interface. Therefore, commands such as `launchpad.exe` or `ifgui.bat` are not supported on that type of Windows Server 2008 installation.

Restriction: These command-line tools do not apply to the Liberty profile. See Administering the Liberty profile from the command prompt instead.

Use the following general steps to run a command:

Procedure

1. Open a system command prompt.
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 information about the `manageprofiles` command topic.

Chapter 5. 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.

gotcha:

- Calling the WebSphere_Ant (WsAnt) scripts outside of the ws_ant launcher is not supported or recommended.
- When you invoke the Ant tool, do not pass empty strings in place of command arguments in ant script. The script will not work in the wsadmin environment. The <arg value> cannot be an empty string, such as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="proj" default="main">
  <taskdef name="wsadmin" classname="com.ibm.websphere.ant.tasks.WsAdmin"/>
  <target name="main">
   <wsadmin,conntype="NONE" lang="jython" failonerror="true" script="&(basedir)/script.ph">
     <arg value="blah" />
     <arg value="" />
   </wsadmin>
   <!-- manaeapp action="blah" variation-number="0" -->
  </target>
</project>
```

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 website.

Procedure

- 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.bat|sh 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 `app_server_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).

  Note: Shipment of Apache Struts 1.1, 1.2.4, and 1.2.7 as optional libraries within WebSphere Application Server is deprecated in Version 7.0.

  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 JavaBeans (EJB) components, and JDBC. For the View, Struts works well with JavaServer Pages (JSP) files, XSLT, or other presentation systems.

Chapter 6. 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.

Procedure

- Use commands to start and stop servers.

Table 16. Commands to start and stop servers.   Run a start or stop command that is appropriate for the target server.

Command
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 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.

Table 17. Opening the administrative console.   Point a Web browser at the console.

Command
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 178 for details, as needed.

  - To launch a scripting client, see Starting the wsadmin scripting client using wsadmin scripting.
  - To learn about all available administrative clients, see Using the administrative clients.
  - For performance monitoring, see Monitoring performance with Tivoli Performance Viewer .

  See the administrator commands that are listed in the **Reference** section of the information center.
- Use troubleshooting tools.

  See Working with troubleshooting tools.

Chapter 7. 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

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.

Procedure

- 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
  - HTTP configuration (see the documentation for your web server)

- Back up your applications.

  This category covers the applications you run using the product. 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

- 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

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 8. 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 269 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.

Procedure

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 260
- "Class-loader isolation policies" on page 261

- "Class-loader modes" on page 263

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 compressed 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.
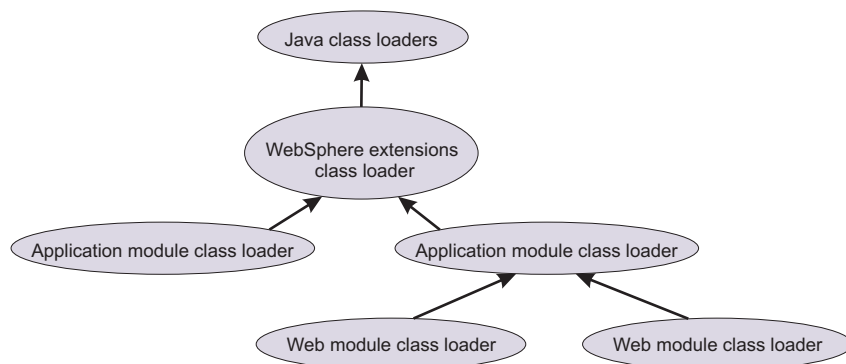
Each class loader is a child of the previous class loader. That is, the application module class loaders are children of the WebSphere extensions class loader, which is a child of the CLASSPATH Java class loader. Whenever a class needs to be loaded, the class loader usually delegates the request to its 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:

Table 18. Class-loader policy descriptions.   Available policies include Application and WAR.

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	By default, web module class loaders load the contents of the `WEB-INF/classes` and `WEB-INF/lib` 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.
	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. **Tip:** The console and the underlying `deployment.xml` 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 `deployment.xml` 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`.

Restriction: 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
```

WebSphere extensions class loader

Classpath:
　　　WebSphere/AppServer/classes
　　　WebSphere/AppServer/lib
　　　WebSphere/AppServer/lib/ext

Application class loader

Classpath:
　　　Ejb1.jar
　　　Dependency1.jar
　　　Ejb2.jar
　　　Dependency2.jar
　　　WAR2.war (WEB-INF/classes, ...)

WAR class loader

　　WAR1.war
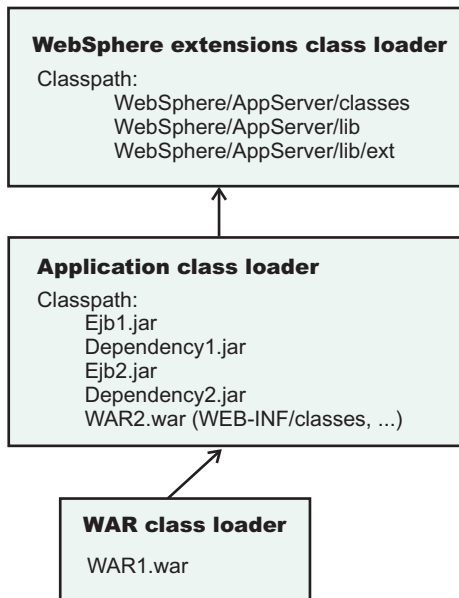
The following example shows that when the application class-loader policy of an application server is set to `Multiple`, each application on the server has its own class loader. An application class loader also loads its web modules if the application WAR class-loader policy is set to `Application`. 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
```

WebSphere extensions class loader

Classpath:
      WebSphere/AppServer/classes
      WebSphere/AppServer/lib
      WebSphere/AppServer/lib/ext

Application class loader
Classpath:
    Ejb1.jar
    Dependency1.jar

Application class loader

Classpath:
    Ejb2.jar
    Dependency2.jar
    WAR2.war (WEB-INF/classes, ...)

WAR class loader
WAR1.war
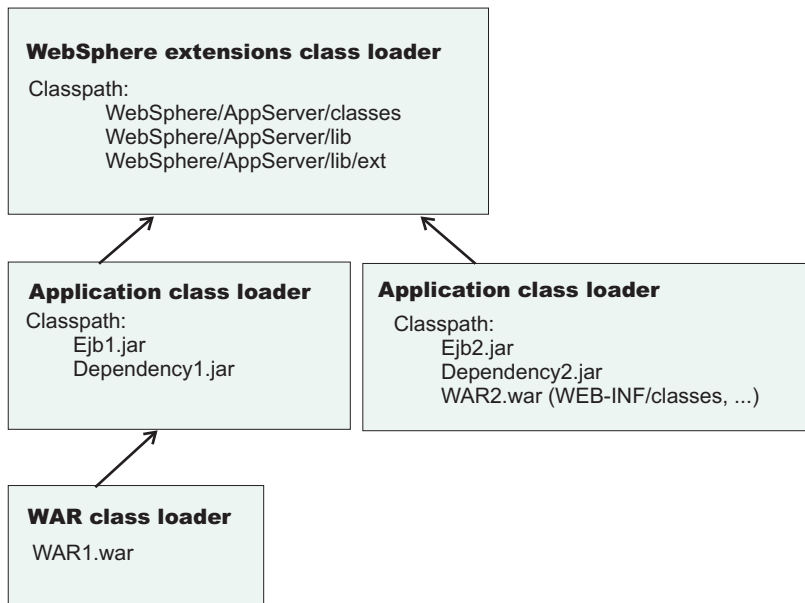
Class-loader modes

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:

Table 19. Class-loader mode descriptions.   Available modes include Parent first and Parent last.

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.

Procedure

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.

Information	Value
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.

Information	Value
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 archive (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.

Attention: 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.

Procedure

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).

   Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:
   - For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
   - For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

     However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

     The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

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 **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 `Classes loaded with local class loader first (parent last)`, your application can override classes contained in the parent class loader. **Note:** Specifying the `Classes loaded with local class loader first (parent last)` value might result in LinkageErrors or ClassCastException messages if you have mixed use of overridden classes and non-overridden classes.

5. 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.

6. 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`, then the application class loader loads the web module contents as well as the enterprise bean (EJB) modules, shared libraries, resource adapter archive (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. You can set the policy on the Class loading and update detection page of an administrative console.

Use the administrative console to configure the application and WAR class loaders.

Attention: 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.

Procedure

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	This option causes the class loader to *prefer classes that are provided by the product* over the classes that exist within the web module. This approach is standard for Development Kit class loaders and WebSphere Application Server class loaders.
Classes loaded with local class loader first	This option causes the class loader to *prefer classes that exist in the web module* over the classes that are provided by the product. If the same class exists in both the product and the web module, the class from the web module is loaded. **Attention:** If you specify the `Classes loaded with local class loader first` value, you might receive LinkageErrors or ClassCastException 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 publications 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

- 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 V7: Understanding Class Loaders, `http://publib-b.boulder.ibm.com/abstracts/redp4581.html?Open`

Programming specifications

- Specifications and API documentation

Chapter 9. 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).

Procedure
- 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 using the console.
- Uninstall enterprise applications by removing them from a monitored directory.
- 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 previously described.

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.

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 enterprise 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.

Procedure
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
    - Drag and drop to a monitored directory
    - 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.

Note:

- You must use either the administrative console or wsadmin scripting to synchronization a node. Of these two options, using the administrative console is the best way to perform this operation. The Nodes panel in the administrative console includes the **Synchronize** operation.

  If you need to use wsadmin scripting to synchronize a node, use the NodeSync mbean's sync() command.

- Do not restart the node agent as part of the synchronize node process. Administration operations, such as node synchronization for application deployment, or updates that take place while the node agent is starting, that are initiated through the node agent, and affect the application servers, fail until the node agent has a chance to discover the application servers.

Next, test the application. For example, point a web browser at the URL for a deployed application. Typically, the URL is `http://hostname:9060/web_module_name`, where *hostname* is your valid web server and 9060 is the default port number. Examine the performance of the application. If the application does not perform as desired, edit the application configuration, then save and test it again.

If your application contains many classes with annotations and takes a long time to deploy, you can reduce annotation searches to speed up deployment. See the topic on reducing annotation searches during application deployment.

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) archive (SAR), web module, or client module on a Version 8.x target under any of the following conditions:
- The module supports Java Platform, Enterprise Edition (Java EE) 6.
- The module calls an 8.x runtime application programming interface (API).
- The module uses an 8.x product feature.

  For example, because support for deployment of application client modules using the administrative console or wsadmin AdminApp commands was added in Version 8.0, you must install a client module using the console or an AdminApp command only to a Version 8.x target.

If a module supports Java 2 Platform, Enterprise Edition (J2EE) 1.4, then you can install the module on a Version 6.x, 7.x or 8.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, 7.x or 8.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, 7.x or 8.x deployment target. Modules that require 6.1.x feature pack functionality can be installed on a 7.x or 8.x deployment target or on a 6.1.x deployment target that has been enabled with that feature pack. Modules that require 7.x feature pack functionality can be installed on a 8.x deployment target or on a 7.x deployment target that has been enabled with that feature pack.

Selecting options such as **Precompile JavaServer Pages files**, **Deploy web services** or **Deploy enterprise beans** during application installation indicates that the application uses features of the current product version. Do not select these options if the targets of the applications are on older version nodes. Use the tools available in the older version, such as JspBatchCompiler, wsgen, or ejbdeploy, to update your application with generated code before deploying your application.

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 or EJB 3.1 module that contains CMP or BMP entity beans. Installation fails when a CMP or BMP entity bean is packaged in an EJB 3.0 or EJB 3.1 module. You can install EJB 2.1 or earlier modules on a 6.x, 7.x or 8.x deployment target.

Installable RAR files

You can install a stand-alone resource adapter (connector) module, or RAR file, developed for a Version 6.0.x product to a 6.x, 7.x or 8.x deployment target. If the module calls a 6.1.x API, then you must install the module on a 6.1.x, 7.x or 8.x deployment target. You must install a module that calls a 7.x API on a 7.x or 8.x deployment target. You must install a module that calls a 8.x API on a 8.x deployment target.

Deployment targets

The following table lists the compatible deployment target versions for various modules. "6.x, 7.x or 8.x" for Deployment target versions indicates that you can deploy the module to a WebSphere Application Server Version 6, 7, or 8 server.

Table 20. Compatible deployment target versions for 6.x, 7.x and 8.x modules.   Deploy modules to compatible deployment target versions.

Module type	Module Java support	Module calls 6.x, 7.x or 8.x runtime APIs or uses 6.x, 7.x or 8.x features?	Client versions that can install module	Deployment target versions
Application, EJB, or web	J2EE 1.3	No	6.x, 7.x or 8.x	6.x, 7.x or 8.x
Application, EJB, or web	J2EE 1.3	Yes	6.x, 7.x or 8.x for 6.x APIs or features 7.x or 8.x for 7.x APIs or features 8.x for 8.x APIs or features	6.x, 7.x or 8.x You must install modules that call 6.1.x runtime APIs or use 6.1.x features on a 6.1.x, 7.x or 8.x deployment target. You can install modules that call 6.0.x runtime APIs or use 6.0.x features on any 6.x, 7.x or 8.x deployment target.
Application, EJB, SAR, or web	J2EE 1.4	Yes or No	6.x, 7.x or 8.x	6.x, 7.x or 8.x
Application, EJB, SAR, or web	Java EE 5	Yes or No	7.x or 8.x	7.x or 8.x
Application, EJB, SAR, or web	Java EE 6	Yes or No	8.x	8.x
Client	Any Java EE version	Yes or No	8.x	8.x
Resource adapter	JCA 1.0	No	6.x, 7.x or 8.x	6.x, 7.x or 8.x

Table 20. Compatible deployment target versions for 6.x, 7.x and 8.x modules (continued).   Deploy modules to compatible deployment target versions.

Module type	Module Java support	Module calls 6.x, 7.x or 8.x runtime APIs or uses 6.x, 7.x or 8.x features?	Client versions that can install module	Deployment target versions
Resource adapter	JCA 1.0	Yes	6.x, 7.x or 8.x	6.x, 7.x or 8.x You must install modules that call 6.1.x runtime APIs on a 6.1.x, 7.x or 8.x deployment target. You can install modules that call 6.0.x runtime APIs on any 6.x, 7.x or 8.x deployment target.
Resource adapter	JCA 1.5	Yes or No	6.x, 7.x or 8.x	6.x, 7.x or 8.x You must install modules that call 6.1.x runtime APIs on a 6.1.x, 7.x or 8.x deployment target. You can install modules that call 6.0.x runtime APIs on any 6.x, 7.x or 8.x deployment target.
Resource adapter	JCA 1.6	Yes or No	JCA 1.6 resource adapters can only be installed on 8.x. Resource adapter archive annotations are not supported on previous WebSphere Application Server releases.	JCA 1.6 resource adapters can only be installed on 8.x. Resource adapter archive annotations are not supported on previous WebSphere Application Server releases.

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 application archive (WAR), Session Initiation Protocol (SIP) archive (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 21. Ways to install application files.   Deploy an application or module using the administrative console, wsadmin, programming, or deployment tools.

Option	Method	Modules	Comments	Starting after install
Administrative console install wizard See topics on installing enterprise application files with the console.	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: • EAR • EJB • WAR • SAR • RAR • Application client	Provides one of the easier ways to install application files. 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.
Monitored directory	Add an EAR file or module to a `dragDropDeployableApps` subdirectory of an application server profile. For base (stand-alone) application servers, the monitored directory is the `dragDropDeployableApps` `/servers/`*server_name* directory of the application server profile.	Files for all of the following modules: • EAR • EJB • WAR • SAR Use this option for drag and drop deployment of Java EE 5.0 and later modules.	Provides one of the easier ways to install applications. You cannot specify bindings during deployment. For applications that require changes to the bindings, install the application using the administrative console install wizard, application properties files, or wsadmin scripts. Monitored directory deployment differs from rapid deployment tools in several ways: • Monitored directory deployment supports deployment to base and network deployment environments. Rapid deployment tools support deployment only to the base environment. • Monitored directory deployment does not start a new daemon. Rapid deployment tools start a separate process. • Monitored directory deployment supports deployment of Java EE 5 and later modules. Rapid deployment tools support assembly of J2EE 1.3 and 1.4 modules, and deployment of all Java EE module versions. • Monitored directory deployment supports use of a properties file to specify deployment options. Rapid deployment tools does not support use of a properties file.	After application or module installation or update, the product starts the application or module automatically.
Application properties files	Create a properties file that specifies to install or update application files. Then, run the wsadmin **applyConfigProperties** command in any of the following ways: • In a script • At a command prompt • By adding the properties file to a `dragDropDeployableApps` `/deploymentProperties` monitored directory	Files for all of the following modules: • EAR • EJB • WAR • SAR	For information on deploying applications using properties files, see the following topics: • Using application properties files to install, update, and delete enterprise application files • Installing enterprise application files by adding properties files to a monitored directory	After application or module installation or update, the product starts the application or module automatically.
wsadmin scripts	Invoke AdminApp object **install** commands in a script or at a command prompt.	Files for all of the following modules: • EAR • EJB • WAR • SAR • RAR • Application client	"Getting started with scripting" in the Using the administrative clients PDF provides an overview of wsadmin.	• Invoke the AdminApp **startApplication** command. • Invoke the **startApplication** method on an ApplicationManager MBean using AdminControl.

Table 21. Ways to install application files (continued).   Deploy an application or module using the administrative console, wsadmin, programming, or deployment tools.

Option	Method	Modules	Comments	Starting after install
Java application programming interfaces	Install programs by completing the steps in Installing an application through programming.	All EAR files	Use Java Management Extensions (JMX) MBeans to install the application. For an overview of Java MBean programming, see Managing applications through programming.	Start the application by calling the `startApplication` method on a proxy.
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: • EJB • WAR • SAR • RAR • Application client The rapid deployment tools do not support the J2EE 1.2 or Java EE 5.0 and later specification levels. Use this option for drag and drop deployment of J2EE 1.3 or 1.4 modules. Unlike the monitored directory option, the rapid deployment tools do not support drag and drop deployment of Java EE 5.0 and later modules.	Rapid deployment tools offer the following advantages: • 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. For a list of ways in which the rapid deployment tools differ from monitored directory deployment, see the monitored directory description in this table.	Use any of the options in this table 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. **Note:** Application installation using JSR-88 was deprecated in WebSphere Application Server Version 8.0. Use another way listed in this table to deploy applications or modules.	All Java EE modules, including EAR files and the following stand-alone modules: • EJB • WAR • SAR • RAR • Application client	• 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.

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
- Monitored directory deployment
- Application properties files
- 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) archive (SAR), or web module.

Note: After you start completing steps in the application installation wizard, 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.

Procedure

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 archive (WAR), SAR, or Java archive (JAR) file for installation.

   Note: 　Windows　 If you attempt to install an application from a remote file system and the Java virtual machine (JVM) is registered as a Microsoft Windows service, the administrative console does not show a mapped network drive. In this scenario, the Windows service runs as a new log on session whose security identifier (SID) is different from the user session. Thus, the service cannot manipulate the mapped network drive, which is established by the login user. To install the application from a remote file system, you must use the **WASService** command to unregister the JVM from the Windows service. For more information, see the documentation about the **WASService** command.
   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 application bindings. 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.x modules. The product assigns container default values during run time. Similarly, for any EJB reference within an EJB 3.x or a Web 2.4 or later 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 288 describes available customization 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 285 provides sample steps.

4. Specify values for installation options as needed.

You can click on a step number to move directly to that page instead of clicking **Next**. The contents of the application or module that you are installing determines which pages are available.

Table 22. Wizard page descriptions. The table describes each wizard page.

Page	Description
Select installation options	On the **Select installation options** page, 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** page, 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** page and your application uses JavaServer Pages (JSP) files, then you can specify JSP compiler options on the **Provide options to compile JSPs** page.
Provide JNDI names for beans	On the **Provide JNDI names for beans** page, 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.x 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** page, you can specify business interface JNDI names for EJB 3.x modules. If you specified a JNDI name for a bean on the **Provide JNDI names for beans** page, do not specify a business interface JNDI name on this page 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. For a no-interface view, the business interface value is an empty string (`""`).
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.
Map EJB references to beans	On the **Map EJB references to beans** page, if your application defines EJB references, you can specify JNDI names for enterprise beans that represent the logical names specified in EJB references. If the EJB reference is from an EJB 3.x, or Web 2.4 or later module, the JNDI name is optional. For earlier modules, each EJB reference defined in the application must be bound to an EJB file. 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.

Table 22. Wizard page descriptions (continued).   The table describes each wizard page.

Page	Description
Map resource references to resources	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 **Finish** on the Summary page.
Map virtual hosts for web modules	If your application uses web modules, for **Map virtual hosts for web modules**, select a virtual host from the list to 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 page.
Map security roles to users or groups	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 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: 1. Select a role and click **Lookup users** or **Lookup groups**. 2. On the Lookup users or groups page 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** page.
Map RunAs roles to users	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 the roles or select individual roles. After selecting a role, enter values for the user name, password, and verify password and click **Apply**.
Ensure all unprotected 1.x methods have the correct level of protection	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.
Bind listeners for message-driven beans	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.
Map default data sources for modules containing 2.x entity beans	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.

Table 22. Wizard page descriptions (continued).   The table describes each wizard page.

Page	Description
Map data sources for all 2.x CMP beans	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** page, for each of the 2.x CMP beans specify a JNDI name and the type of resource authorization for data sources to be used.
	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 a default data source for the EJB module and a data source for individual CMP beans are not specified, then a validation error is displayed after you click **Finish** and installation is canceled.
Ensure all unprotected 2.x methods have the correct level of protection	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** page, specify whether you want to assign a specific role to the unprotected methods, add the methods to the exclude list, or mark them as deselected. Methods added to the exclude list are marked as uncallable. For methods marked deselected no authorization check is performed before their invocation.
Provide options to perform the EJB Deploy	If the **Deploy enterprise beans** setting is enabled on the **Select installation options** page, then you can specify options for the EJB deployment tool on the **Provide options to perform the EJB Deploy** page. On this page, you can specify extra class paths, RMIC options, database types, and database schema names to be used while running the EJB deployment tool.
	You can specify the EJB deployment tool options on this page 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.x modules.
Map shared libraries	On the **Shared library references** and **Shared library mapping** pages, 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.
Map shared library relationships	On the **Map shared library relationships** page, 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 for **Business-level application name** on the **Select installation options** page.
Provide JSP reloading options for web modules	If your application uses web modules, for **Provide JSP reloading options for web modules**, configure the class reloading of JavaServer Pages (JSP) files.
Map context roots for web modules	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.
	The product does not include web modules from annotations on this page.
Initialize parameters for servlets	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.
	This page shows servlets from the module XML deployment descriptor. Servlet deployment information from annotations is not available on this page.

Table 22. Wizard page descriptions (continued).   The table describes each wizard page.

Page	Description
Map environment entries for EJB modules	If your application uses EJB modules, for **Map environment entries for EJB modules**, configure the environment entries of EJB modules such as entity, session, or message driven beans.
Map environment entries for client modules	If you are deploying one or more application client modules, for **Map environment entries for client modules**, configure the environment entries of client modules that are deployed as JAR files. To view the Map environment entries for client modules page, select the **Deploy client modules** option on the Select installation options page.
Map environment entries for web modules	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.
Map environment entries for application level	If your application defines one or more environment entries, for **Map environment entries for application level**, configure the environment entries of applications that are deployed as EAR files.
Map resource environment entry references to resources	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.
Correct use of system identity	If your application defines **Run-As Identity** as *System Identity*, for **Correct use of system identity**, you can optionally change it to *Run-As role* and specify a user name and password for the Run As role specified. Selecting *System Identity* 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.
Correct isolation levels for all resource references	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.
Map JASPI Provider	On the **Map JASPI Provider** page, if your application has web modules, you can specify values to override the JASPI settings from the global or domain security configuration. By default, an application inherits the JASPI settings defined in the WebSphere Application Server global or domain security configuration, and web modules inherit the application setting.
Bind message destination references to administered objects	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. If the message destination reference is from an EJB 3.0 or later module, then the JNDI name is optional and the run time provides a container default value. **Attention:** 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.
Provide JNDI names for JCA objects	If your application contains an embedded `.rar` file, for **Provide JNDI names for JCA objects**, specify the name and JNDI name of each JCA connection factory, administered object and activation specification.

Table 22. Wizard page descriptions (continued).   The table describes each wizard page.

Page	Description
Bind J2C activationspecs to destination JNDI names	If your application contains an embedded `.rar` file, its activationSpec property has the value `Destination`, and its introspected type is `javax.jms.Destination`, for **Bind J2C activationspecs to destination JNDI names**, specify the jndiName value for each activation bound to it.
Select current backend ID	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. For information about backend databases, see topics on the EJB deployment tool. This step is not shown if the **Deploy enterprise beans** setting is enabled on the **Select installation options** page and if a database type other than `None` is specified on the **Provide options to perform the EJB Deploy** page.
Metadata for modules	If your application has EJB 3.x or Web 2.5 modules, you can lock deployment descriptors for one or more of the EJB 3.x or Web 2.5 modules. 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.
Provide options to perform the web services deployment	If the **Deploy web services** setting is enabled on the **Select installation options** page and your application uses web services, then you can specify `wsdeploy` command options on the **Provide options to perform the web services deployment** page. For information about this page, refer to descriptions of the `wsdeploy -cp` and `-jardir` options.
Display module build ID	If the `MANIFEST.MF` file of a module in an enterprise application specifies a build identifer, this page shows the build identifier of the module.

5. On the Summary page, 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 page.
   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** page 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.

For example, click the **Save** link in the application installation messages.

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 page displays again. Click **Applications** > **Application Types** > **WebSphere enterprise applications** to exit the page 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.

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 from the Samples information center.

Procedure

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 in the sample_apps directory.

   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** page, 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 page, 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 an application server.

Before you begin

Download and extract the JaxWSServicesSample sample application. Ensure that your product installation has a Version 7.x or later application server onto which you can install the Web Services Sample.

About this task

The `JaxWSServicesSamples.ear` enterprise application and supporting Java archives (JAR) files are located in the `installableApps` directory within the JaxWSServicesSamples sample application.

This topic describes how to install and start the `JaxWSServicesSamples.ear` enterprise application using an administrative console.

Procedure

1. Click **Applications** > **New Application** > **New Enterprise Application** in the console navigation tree.
2. On the first Preparing for the 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.

        `.../installableApps/JaxWSServicesSamples.ear`
    b. Click **Next**.
3. On the second Preparing for the 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 page until you reach the Summary page.

    Do not go directly from Step 1 to the Summary page. You must click **Next** on each page 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 page, 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 Port descriptions table lists the important ports.

Table 23. Port descriptions. Use the `WC_defaulthost` port in the URL to test the sample.

Port name	Description
`WC_adminhost`	Port used to open an unsecure administrative console in the URL `http://host_name:administrative_port/ibm/console`
`WC_adminhost_secure`	Port used to open a secure administrative console in the URL `http://host_name:administrative_port/ibm/console`
`WC_defaulthost`	Port used to test running applications in the URL `http://host_name:server_port/servlet_name`
`WC_defaulthost_secure`	Port used to securely test running applications in the URL `http://host_name:server_port/servlet_name`

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 choose 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.

Table 24. Installation option descriptions.   You can select a Fast Path or select to see all installation options and parameters.

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.

Table 25. Binding option descriptions.   You can use merged, new, or existing bindings.

Option	Description
Merge new and existing bindings	The binding information from the updated application or modules is preferred over the corresponding binding information from the installed version. If any element of the binding is missing in the updated version, the corresponding element from the installed version is used. If both the installed and the updated application or module does not have a binding value, the default value is used. The product assigns a default value only if you select the **Generate default bindings** option.
Use new bindings	The binding information in the updated application or module is used. The binding information from the updated version of the application or module is preferred over the corresponding binding information in the installed version. The binding information from the installed version of the application or module is ignored.
Use existing bindings	The binding information from the installed version of the application or module is preferred over the corresponding binding information from the updated version. If any element of the binding information does not exist in the installed version, the element from the updated version is used. That is, bindings from the updated version of the application or module are ignored if a binding exists in the installed version. Otherwise, the new bindings are honored and not ignored.

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 page, verify the cell, node, and server on which the application is installed.

transition: 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 or later modules. The product assigns container default values during run time. Similarly, for any EJB reference within an EJB 3.0, EJB 3.1, 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 or later modules. The product does not generate default values for enterprise beans in an EJB 3.0 or later 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, EJB 3.1, 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 or 6 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 and 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-link`Name. Otherwise, the JNDI name is set to `eis/message-destination-ref`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.
- 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>
```

> **Remember:** 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>
```

> **Restriction:** 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/somAO</jndi-name>
        </message-destination-ref-binding>
       </message-destination-ref-bindings>
      </ejb-binding>
     </ejb-bindings>
    </ejb-jar-binding>
   </module-bindings>
 </dfltbndngs>
```

> **Restriction:** 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>
```

Restriction: 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>
        <activationspec-jndi-name>activationSpecJNDI</activationspec-jndi-name>
      </ejb-binding>
     </ejb-bindings>
   </ejb-jar-binding>
  </module-bindings>
 </dfltbndngs>
```

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 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>
          <listener-port>yourMdbListPort</listener-port>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

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 dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
   <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>
</dfltbndngs>
```

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 page 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 page, 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 page 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 8.5 deployment target.

If you select **Precompile JavaServer Pages files** and try installing your application onto an earlier deployment target such as Version 7, 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.

Information	Value
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 computers, 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.

  Important: 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.

Information	Value
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.

Information	Value
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.

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 binary configuration** field is the same as the **Use configuration information in binary** setting on an Application binaries page.

Information	Value
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 JavaBeans (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.

If an EJB module is packaged in a web archive (WAR), you do not need to enable this setting.

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.x modules.

For this option, install only onto a Version 8.5 deployment target.

If you select **Deploy enterprise beans** and try installing your application onto an earlier deployment target such as Version 7, 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** page, 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** page.

Enabling this setting might cause the installation program to run for several minutes.

Information	Value
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:

Table 26. Characters that you cannot use in a name.   The product does not support these characters in a name.

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.

Information	Value
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.

Information	Value
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 a Class loading and update detection page.

Information	Value
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.xmi`) 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.

Information	Value
Data type	Integer
Units	Seconds
Default	3

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

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 8.5 deployment target.

If you select **Deploy web services** and try installing your application onto an earlier deployment target, 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.

Information	Value
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.

Information	Value
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`.

This setting affects installation of enhanced EAR files. An enhanced EAR file results when you export an installed application.

When `false`, an enhanced EAR file is installed like any other application and the product ignores its embedded configuration.

If you exported the application from a cell other than the current cell and did not specify the `$(CELL)` variable for **Directory to install application** when first installing the application, deselect this setting (`false`) to expand the enhanced EAR file in the *profile_root*/installedApps/*current_cell_name* directory. Otherwise, if this setting is selected (`true`), the enhanced EAR file is expanded in the *profile_root*/installedApps/*original_cell_name* directory, where *original_cell_name* is the cell on which the application was first installed. If you specified the `$(CELL)` variable for **Directory to install application** when you first installed the application, installation expands the enhanced EAR file in the *profile_root*/installedApps/*current_cell_name* directory.

Information	Value
Data type	Boolean
Default	false (deselected)

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.

Table 27. File permission string sets for list options.   Select a list option or specify a file permission string in the text field.

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`

Table 27. File permission string sets for list options (continued).   Select a list option or specify a file permission string in the text field.

Multiple-selection list option	File permission string set
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.

best-practices: 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:

Table 28. Example URIs for file permission operations.   Results are shown following this table.

Number	Example URL
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

best-practices: Regardless of the operation system, always use a forward slash (/) as a file path separator in file patterns.

█Windows█ You cannot unset read permission on a file on Windows operating systems. With POSIX style permission bits, the bit for denoting `readable` on a file is 4, `writable` is 2, and `executable` is 1. Thus, permission of a file on a Windows operating system is either 5 or 7. Also, in POSIX style there are `user`, `group` and `world` permissions. You can only set the `user` permission for a file on Windows operating systems. The `group` and `world` permission bits are ignored.

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.

Information	Value
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.

Information	Value
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. If a business-level application with the name of your enterprise application exists already, the product does not create a new business-level application; it adds your enterprise application as a composition unit to that existing business-level application.

If you need to use the Shared library relationship and mapping settings page to specify dependency relationships on existing shared libraries in the business-level application, select the business-level application name from the drop-down list. No shared libraries are shown in the page if you choose to create a new business-level application and a business-level application with the default name exists already.

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.

Information	Value
Data type	String

Information	Value
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.

Information	Value
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.

Information	Value
Data type	Boolean
Default	false

Deploy client modules

Specifies whether to deploy client modules.

Select this option (set to `true`) if the file to deploy has one or more client modules and you want to configure environment entries for the client modules. Also select this option to configure resources such as EJB references, resource references, resource environment references, or message destination references. Selecting this option enables you to view the Map environment entries for client modules page. If you are deploying the client modules to a federated node of a deployment manager (**Federated**) or to an application server (**Server Deployed**), select this option and set **Client deployment mode** to the appropriate value for the deployment target, **Federated** or **Server Deployed**.

If you select this option, install the client modules only onto a Version 8.0 or later deployment target.

Information	Value
Data type	Boolean
Default	false

Client deployment mode

Specifies whether to deploy client modules to an isolated deployment target (**Isolated**), a federated node of a deployment manager (**Federated**), or an application server (**Server Deployed**).

The choice of client deployment mode affects how `java:` lookups are handled. All Java URL name spaces (global, application, module, and component) are local in isolated client processes. The name spaces reside on a server in federated and server deployed client processes. The server chosen as a target for a client module determines where those name spaces are created. All `java:` lookups for federated or server deployed client modules are directed to the target server. The client module does not actually run in the target server. Multiple instances of the same client module will all share the component name space in the **Federated** and **Server Deployed** modes. Choosing the **Federated** mode is simply a declaration of intent to launch the client module using Java Network Launching Protocol (JNLP), but the Java Naming and Directory Interface (JNDI) mechanics of federated and server deployed modes are the same.

Information	Value
Data type	String
Default	Isolated

Validate schema

Specifies whether to validate the deployment descriptors against published Java EE deployment descriptor schemas. When this option is selected, the product analyzes each deployment descriptor to determine the Java EE specification version for the deployment descriptor, selects the appropriate schema, and then checks the deployment descriptor against the Java EE deployment descriptor schema. Validation errors result in error messages.

A Java EE deployment descriptor schema is also known as a *DTD*.

If you select this option, install your application or module only onto a Version 8.0 or later deployment target.

Information	Value
Data type	Boolean
Default	false

Manage modules settings

Use this page 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 page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Manage modules**. This page is the similar to the Map modules to servers page on the application installation and update wizards.

On this page, 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 8.x features on a Version 7.x or 6.x target server. Similarly, you cannot install modules that use Version 7.x features on a Version 6.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**.

Attention: 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 page from a console enterprise application page for an already installed application, you can also use this page 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:

Table 29. Button descriptions.   Use the buttons to manage modules.

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) archive (SAR), web module, or client module on a Version 8.x target under any of the following conditions:
- The module supports Java Platform, Enterprise Edition (Java EE) 6.
- The module calls an 8.x runtime application programming interface (API).
- The module uses an 8.x product feature.

You must install an application, EJB, SAR, or web module on a Version 8.x or 7.x target under any of the following conditions:
- The module supports Java EE 5.
- The module calls a 7.x runtime 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, 7.x or 8.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, 7.x or 8.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 or 8.x deployment target.

You can install an application or module developed for a Version 5.x product on any 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 page 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.

Client module settings
Use this page to configure a deployed client module.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Manage modules** > *client_module_name*. This page is viewable only if the selected application contains a client module and the client deployment mode is a value other than `isolated`.

URI
Specifies the location of the client module relative to the root of the application.

Alternate deployment descriptor
Specifies the alternate deployment descriptor for the module as defined in the application deployment descriptor according to the Java Platform, Enterprise Edition (Java EE) specification.

Client module property settings
Use this page to configure the deployment mode of a deployed client module.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Client module deployment mode**. This page is viewable only if the selected application contains a client module.

Client module deployment mode
Specifies whether to deploy client modules to an isolated deployment target (**Isolated**) or an application server (**Server Deployed**).

Information	Value
Data type	String
Default	Isolated

Provide options to compile JavaServer Pages settings

Use this page to specify options to be used by the JavaServer Pages (JSP) compiler.

This administrative console page is a step in the application installation and update wizards. To view this page, you must select **Precompile JavaServer Pages files** on the **Select installations options** page. Thus, to view this page, click **Applications** > **New Application** > **New Enterprise Application** > *application_path* > **Next** > **Detailed - Show me all installation options and parameters** > **Next** > **Next** or **Continue** > **Precompile JavaServer Pages files** > **Next** > **Step: Provide options to compile JSPs**.

You can specify the JSP compiler options on this page only when installing or updating an application that contains web modules. After the application is installed, you must edit the JSP engine configuration parameters of a web module `WEB-INF/ibm-web-ext.xmi` file to change its JSP compiler options.

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Web module

Specifies the name of a module within the application.

URI

Specifies the location of the module relative to the root of the application (EAR file).

JSP class path

Specifies a temporary class path for the JSP compiler to use when compiling JSP files during application installation. This class path is not saved when the application installation is complete and is not used when the application is running. This class path is used only to identify resources outside of the application which are necessary for JSP compilation and which will be identified by other means (such as shared libraries) after the application is installed. In network deployment configurations, this class path is specific to the deployment manager machine.

To specify that multiple web modules use the same class path:
1. In the list of web modules, select the **Select** check box beside each web module that you want to use a particular class path.
2. Expand **Apply Multiple Mappings**.
3. Specify the desired class path.
4. Click **Apply**.

Use full package names

Specifies whether the JSP engine generates and loads JSP classes using full package names.

When full package names are used, precompiled JSP class files can be configured as servlets in the `web.xml` file, without having to use the `jsp-file` attribute. When full package names are not used, all JSP classes are generated in the same package, which has the benefit of smaller file-system paths.

When the options useFullPackageNames and disableJspRuntimeCompilation are both `true`, a single class loader is used to load all JSP classes, even if the JSP files are not configured as servlets in the `web.xml` file.

This option is the same as the useFullPackageNames JSP engine parameter.

JDK source level

Specifies the source level at which the Java compiler compiles JSP Java sources. Valid values are `13`, `14`, and `15`. The default value is `13` for pre-Java EE 5 web modules, which specifies source level 1.3 and `15` for Java EE 5 and later web modules.

Disable JSP runtime compilation

Specifies whether a JSP file should never be translated or compiled at run time, even when a `.class` file does not exist.

When this option is set to `true`, the JSP engine does not translate and compile JSP files at run time; the JSP engine loads only precompiled class files. JSP source files do not need to be present in order to load class files. You can install an application without JSP source, but the application must have precompiled class files.

For a single web application class loader to load all JSP classes, this compiler option and the **Use full package names** option both must be set to `true`.

This option is the same as the disableJspRuntimeCompilation JSP engine parameter.

EJB JNDI names for beans

Use this page to view and modify the Java Naming and Directory Interface (JNDI) names of non-message-driven enterprise beans in your application or module.

If your application uses Enterprise JavaBeans (EJB) 2.1 and earlier modules, 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 EJB 2.1 and earlier enterprise bean defined in the application. For example, for the EJB module MyBean.jar, specify MyBean.

The JNDI name for an EJB module can be used for both EJB 3.x modules and pre-EJB 3.0 modules. For a pre-EJB 3.0 module, you need to provide a JNDI name for the bean. For an EJB 3.x module, you have three options

- Provide no JNDI names at all
- Select the radio button to provide a JNDI name for the bean, or
- Select the radio button to provide local or remote home JNDI names.

If no JNDI name is provided, the run time provides a default value. If JNDI name for the bean is provided, you cannot provide any JNDI name for business interface in the Provide JNDI names for business interfaces panel.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > **application** > **EJB JNDI names**.

Attention: 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.

Module
Specifies the name of the Enterprise JavaBeans module used by your application.

Bean
Specifies the name of an enterprise bean that is contained by the module.

URI
The Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.

Target Resource JNDI name
Specifies the Java Naming and Directory Interface (JNDI) name of the enterprise bean.

This is a data entry field. To modify the JNDI name bound to this bean, type the new name in this field, then select **OK**.

Information	Value
Data type	String

Bind EJB business settings

Use this administrative console page to specify Java Naming and Directory (JNDI) name bindings for each enterprise bean with a business interface in an EJB module. Each enterprise bean with a business interface in an EJB module must be bound to a JNDI name. For any business interface that does not provide a JNDI name, or if its bean does not provide a JNDI name, a default binding name is provided. If its bean provides a JNDI name, the default JNDI name for the business interface is provided on top of its bean JNDI name by appending the package-qualified class name of the interface.

If you specify the JNDI name for a bean in the Provide JNDI names for beans page, do not specify any business interface JNDI name in this page for the same bean. If you do not specify the JNDI name for a bean in the Provide JNDI names for beans page, you can optionally specify a business interface JNDI name. If you do not specify a business interface JNDI name, the run time provides a container default.

To view this page in the administrative console, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Bind EJB business**.

Attention: 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.

Module
Specifies the EJB module that contains the enterprise beans that bind to the JNDI name.

Bean
Specifies the enterprise bean that binds to the JNDI name.

URI

The Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.

Business Interface

Specifies the enterprise bean business interface in an EJB module.

For a no-interface view, the business interface value is an empty string ("").

JNDI Name

Specifies the JNDI name associated with the enterprise bean business interface in an EJB module.

Map default data sources for modules containing 1.x entity beans

Use this page to set the default data source mapping for EJB modules that contain 1.x container-managed persistence (CMP) beans. Unless you configure individual data sources for your 1.x CMP beans, this default mapping applies to all beans within the module.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Map default data sources for modules containing 1.x entity beans**.

Guidelines for using this administrative console page:

- The page displays a table that depicts the EJB modules in your application that contain 1.x CMP beans.
- Each table row corresponds to a module. A row shows the JNDI name of the data source mapping target of the EJB module *only* if you bound them together during application assembly. For every data source that is displayed, you see the corresponding security configuration.
- To set your default data source mappings:
  1. Select a row. Be aware that if you check multiple rows on this page, the data source mapping target that you select in step 2 applies to all of those EJB modules.
  2. Click **Browse** to select a data source from the new page that is displayed, the Available Resources page. The Available Resources page shows all data sources that are available mapping targets for your EJB modules.
  3. Click **Apply**. The console displays the 1.x entity bean data sources page again. In the rows that you previously selected, you now see the JNDI name of the new resource mapping target.
  4. *Before* you click **OK** to save your new configuration, set the security parameters for the data source. Use the following steps.
- To specify security settings for the default data source:
  1. Select a row. Be aware that if you check multiple rows on this page, the security settings that you select later apply to all of those data sources.
  2. Type in a user name and password that comprise the authentication alias for signing on to the data source. If these entries are not listed in the application Java Platform, Enterprise Edition (Java EE) Connector (J2C) authentication data list, you must input them into the list after saving your settings on this page. Read the information center topic on managing Java EE Connector Architecture authentication data entries for more information.
  3. Click **Apply** that immediately follows the user name and password input fields.
- Repeat all of the previous steps as necessary.
- Click **OK** to save your work.

Select

Select the check boxes of the rows that you want to edit.

EJB Module

The name of the module that contains the 1.x enterprise beans.

URI

Specifies location of the module relative to the root of the application EAR file.

JNDI name

The Java Naming and Directory Interface (JNDI) name of the default data source for the EJB module.

Information	Value
Data type	String

User name

The user name and password that comprise the authentication alias for securing the data source.

EJB references

Use this page to view and modify the Enterprise JavaBeans (EJB) references to the enterprise beans. References are logical names used to locate external resources for enterprise applications. References are defined in the application's deployment descriptor file. At deployment, the references are bound to the physical location (global Java Naming and Directory Interface (JNDI) name) of the resource in the target operational environment.

If your application defines EJB references, for **Map EJB references to beans**, specify JNDI names for enterprise beans that represent the logical names that are specified in EJB references. Each EJB reference defined in the application must be bound to an EJB file before clicking Finish in the Summary panel.

If the EJB reference is from an EJB 3.x, Web 2.4, Web 2.5, or Client 5.0 module, the JNDI name is optional. If the **Allow EJB reference targets to resolved automatically** option is enabled, the JNDI name is optional for all modules. The runtime provides a container default or automatically resolves the EJB reference if a binding is not provided.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > **application_name** > **EJB references**.

Values are displayed for Lookup name and EJB Link if they are configured in the application. Only one of these values is allowed. If both are set, the value must be overridden by a target resource JNDI name.

Attention: 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.

Module

Specifies the name of the Enterprise JavaBeans module used by your application.

Bean

Specifies the name of an enterprise bean that is contained by the module.

URI

Specifies location of the module relative to the root of the application EAR file.

Resource Reference

Specifies the name of the EJB reference that is used in the enterprise bean, if applicable, and declared in the deployment descriptor of the application module.

Class

Specifies the name of a Java class associated with this enterprise bean.

Target Resource JNDI Name

Specifies the JNDI name of the enterprise bean.

This is a data entry field. To modify the JNDI name bound to this bean, type the new name in this field, then select **OK**.

Information	Value
Data type	String

Resource references

Use this page to designate how the resource references of application modules map to the actual resources that are configured for the application.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Resource references**.

You can also view this page during the **Map resource references to resources** step when you install an application.

- If your application uses any of the following resource types, you can set or reset their mapping configurations:
    - Default messaging JMS queues destinations
    - Default messaging JMS topic destinations
    - Data source
    - Generic JMS connection factory
    - Mail session
    - J2C connection factory
    - JMS queue connection factory for the JMS provider of WebSphere MQ
    - JMS queue destination for WebSphere MQ
    - JMS topic connection factory for WebSphere MQ
    - JMS topic destination for WebSphere MQ
    - Unified JMS connection factory for WebSphere MQ
    - URL configuration
- The page is composed of sections that correspond to each applicable resource type. Each section heading is the class name for the resource. If your application contains only one applicable resource type, you see only one section.
- Each section contains a table. Each table row depicts a resource reference within a specific module of your application.
- The rows contain the JNDI names of resource mapping targets for your references *only* if you bound them together during application assembly. You can modify those bindings on this administrative console page.
- To set your mappings:
    1. Select a row. If you want to apply the same mapping to multiple rows, complete the steps in the section, Set multiple JNDI names.

2. Click **Browse** to view a new page listing of all resources that are available mapping targets for your application references.

3. Select a resource and click **Apply**. The console displays the Resource references page again. The JNDI name of the selected resource mapping displays in the Target Resource JNDI Name field.

4. Repeat the previous steps as necessary.

5. If you are editing the resource references of an existing enterprise application, click **OK**. You now return to the general configuration page for your enterprise application. If you are installing the application and have completed the **Map resource references to resources** step, continue to the next step.

- **For data sources and connection factories:** Sections for these resource types contain an additional set of steps for modifying your security settings. Use the last column in the displayed table to view the authorization type for each resource configuration per application module. You can modify the corresponding authentication method only if the authorization type is container. Container-managed authorization indicates that the product performs signon to the resource rather than the enterprise bean code. The reconfiguring process differs slightly for each authentication method option:

  – When you want to assign no authentication method to a resource:

    1. Determine which resource configurations to designate with no authentication method.

    2. Select the appropriate table rows.

    3. Click **Modify Resource Authentication Method** and select **None** from the authentication method options that are displayed above the table.

    4. Click **Apply**.

  – When you want to assign the WebSphere Application Server DefaultPrincipalMapping login configuration to a resource:

    1. You must apply this option to each resource individually if you want to designate different authentication data aliases. See the topic, J2EE connector security, for more information about the default mapping configuration.

    2. Select the appropriate table rows.

    3. Click **Modify Resource Authentication Method** and select **Use default method** from the list of authentication method options that are displayed above the table.

    4. Select an authentication data entry or alias from the list.

    5. Click **Apply**.

  – When you want to assign a trusted context to a resource:

    1. You must have a data source that is running at least DB2 Version 9.1 for z/OS, and the data source must have trusted context enabled.

    2. You must have a data source server that is running at least DB2 Version 9.1 for z/OS, and the data source must have trusted context enabled.

    3. Select the appropriate table rows that have trusted context enabled.

    4. Click **Modify Resource Authentication Method** and select **Use trusted connections** from the authentication method options that are displayed above the table.

    5. Select an authentication alias from the list that matches an alias that is already defined in the DB2 data source. If you do not have an alias defined that is suitable, you must define a new alias.

    6. Click **Apply**.

    7. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

  – When you want to assign a custom Java Authentication and Authorization Service (JAAS) login configuration to a resource:

    1. See the topic, J2EE connector security, for more information about custom JAAS login configurations.

2. Select the appropriate table row.

3. Click **Modify Resource Authentication Method** and select **Use custom login configuration** from the authentication method options that are displayed above the table.

4. Select an application login configuration from the list.

5. Click **Apply**.

6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

Set multiple JNDI names

Use this option to set the same JNDI name on multiple resources with one operation.

Click **Set multiple JNDI names** to display a menu of JNDI names. If you make a selection from this list, it is applied to the **Target Resource JNDI Name** field of all the selected rows of the table.

Modify Resource Authentication Method

Use this panel to toggle the display of a panel above the table rows.

This use of this panel is described in the **For data sources and connection factories** section.

Extended Properties

Use this panel to set additional properties on the selected resource.

Select a single table row and click **Extended Properties** to set additional properties on the selected resource. For more details on using this function, see the documentation on extending DB2 data source definitions at the application level.

Select

Select the check boxes of the rows that you want to edit.

Module

The name of a module in the application.

Bean

The name of an enterprise bean that is contained by the module.

URI

Specifies location of the module relative to the root of the application EAR file.

Resource Reference

The name of a resource reference that is used in the enterprise bean, if applicable, and is declared in the deployment descriptor of the application module.

Target Resource JNDI name

The Java Naming and Directory Interface (JNDI) name of the resource that is the mapping target of the resource reference.

Information	Value
Data type	String

Login configuration

This column applies to data sources and connection factories only and refers to the authorization type and the authentication method for securing the resource.

Virtual hosts settings

Use this page 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 page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Virtual hosts**. This page is the same as the **Map virtual hosts for web modules** page on the application installation and update wizards.

On this page, 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**.

Attention: 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.

Security role to user or group mapping

Use this page to specify the users and groups that are mapped to the security roles that are used with the enterprise application.

To view this administrative console page, click **Applications > Application types > WebSphere enterprise applications >**application_name. Under Detail Properties, click **Security role to user/group mapping**.

Table 30. User and group mapping. User and group mapping.

Button	Resulting action
Map Users	Lists the users that are mapped to the specified role within this application. If trusted realms are configured, a drop-down list of realms to search is displayed. Users from the non-default realm are displayed as user@realm
Map Groups	Lists the groups that are mapped to this specified role within this application. If trusted realms are configured, a drop-down list of realms to search is displayed. Users from the non-default realm are displayed as user@realm
Map Special Subjects	This choice appears if multiple realms are being used. It enables you to map any of the following Special Subjects to a selected role: • **All authenticated in application realm**: All authenticated users that are in the applications realm, which specifies whether to map all of the authenticated users to a specified role. When you map all authenticated users to a specified role, all of the valid users in the current registry who have been authenticated can access resources that are protected by this role. This selection also applies to all authenticated users regardless of the realm. • **Everyone**: map everyone to the selected role. When you map everyone to a role, anyone can access the resources that are protected by this role and, essentially, there is no security. • **None**: Do not map anyone to the selected role **Attention:** • If the secured realm cannot be reached, the left list is replaced with 3 text fields (that is, `name,` `realm,` `and uid`). You can add the user when the secured realm is not available. It is not possible to map two subjects to the same role in this release of WebSphere Application Server.

Role

Lists the specific capabilities to a user. Role privileges give users and groups permission to run as specified.

For example, you might map the user Joe to the administrator role, which enables user Joe to perform all of the tasks associated with the administrator role.

The authorization policy is only enforced when global security is enabled.

Mapped users

Lists the users that are mapped to the specified role within this application.

Special subjects

Lists which special subjects are mapped to the security role when an application uses multiple realms.

Mapped groups

Lists the groups that are mapped to this specified role within this application.

JASPI authentication enablement for applications

Use this page to enable or disable Java Authentication SPI (JASPI) authentication for an application or web module, and to specify the name of a JASPI authentication provider to be used for authenticating messages for the application or web module.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications**. Select an application, and under Detail Properties, select **JASPI provider**.

Select JASPI provider

Select to indicate the web modules in the application that you wish to specify or to override the default JASPI authentication settings for.

Select one of the JASPI provider names to choose a provider to use to perform authentication of web requests for the selected Web module or the application.

To specify how JASPI authentication is performed for the selected web module or the application, choose one of the following:

Do not use JASPI
> Select to disable JASPI authentication for the selected web module or for the application.

Inherit JASPI provider
> Select to inherit the JASPI authentication settings from default values in the cell or domain security configuration, as appropriate.
>
> When Inherit JASPI provider is selected for a web module, JASPI authentication settings for the selected module are the settings that are specified for the application.
>
> When Inherit JASPI provider is selected for the application, JASPI authentication settings are the settings that are specified in the appropriate cell or domain security configuration.

Provider name
> When a specific provider name is selected, that provider is used to perform authentication of web requests for the selected application or web module.

If JASPI authentication is enabled, and a specific provider name is not specified, then the default provider name is used. For more information, read about configuring a new JASPI authentication provider using the administrative console.

If JASPI authentication is disabled, or if no default provider is selected, JASPI authentication is not performed. Web authentication is then performed according to another authentication mechanism as selected in the cell or domain security configuration.

User RunAs collection

Use this page to map a specified user identity and password to a RunAs role. This panel enables you to specify application-specific privileges for individual users to run specific tasks using another user identity.

To view this administrative console page, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name*.
2. Under Detail properties, click **User runAs roles**.

The enterprise beans that you install contain predefined RunAs roles. RunAs roles are used by enterprise beans that need to run as a particular role for recognition while interacting with another enterprise bean.

Username

Specifies a user name for the RunAs role user.

This user already maps to the role specified in the Mapping users and groups to roles panel. You can map the user to its appropriate role by either mapping the user to that role directly or mapping a group that contains the user to that role. After you specify the user name and password for the user and select a RunAs role, click **Apply**.

Note:

The use of the username field is dependent on whether system authorization facility (SAF) delegation is enabled or disabled.

- SAF delegation is enabled. The username field is NOT used.
- SAF delegation is disabled. The username field is used.

Information	Value
Data type:	String

Password

Specifies the password for the RunAs user.

Note:

The use of the password field is dependent on whether system authorization facility (SAF) delegation is enabled or disabled.

- SAF delegation is enabled. The password field is NOT used.
- SAF delegation is disabled. The password field is used.

Information	Value
Data type:	String

Role

Maps specific capabilities to a user.

The authorization policy is only enforced when global security is enabled.

Ensure all unprotected 1.x methods have the correct level of protection

Use this page to verify that the unprotected Enterprise JavaBeans (EJB) Version 1.*x* methods have the correct level of protection before you map users to roles.

This administrative console page is displayed during the application deployment process. To access the administrative console page, click **Application** > **New application** > **New Enterprise Application**. The page is displayed as **Ensure all unprotected 1.x methods have the correct level of protection** in the application deployment steps. On this administrative console page, you can specify whether users can access specific EJB modules.

EJB module

Specifies the EJB module name.

URI

Specifies the Uniform Resource Identifier (URI) that is used to locate the Java archive (JAR) file for the EJB module.

Deny all access

Select this option to protect this EJB module by making it inaccessible to users regardless of their access permissions.

Information	Value
Default:	Cleared

Bind listeners for message-driven beans settings

Use this page to specify bindings for message-driven beans in your application or module.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Message Driven Bean listener bindings**. This page is the same as the **Bind listeners for message-driven beans** page 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 either of the following Java Message Service (JMS) providers:
* WebSphere MQ messaging provider
* Generic messaging provider

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** page.
* 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. Complete one of the following steps:
   * Specify a listener port name.
   * Select a target resource JNDI name for an activation specification. Optionally specify the following parameters:
     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.
     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.

4. Click **Apply**.

5. Click **OK** or **Next**.

Module

Specifies the name of the module that contains the enterprise bean.

Bean

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.

Listener 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`.

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.

- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Map data sources for all 2.x CMP beans

Use this page to set the default data source mapping for EJB modules that contain 2.x container-managed persistence (CMP) beans. Unless you configure individual data sources for your 2.x CMP beans, this default mapping applies to all beans within the module.

To view this administrative console panel, click **Applications** > **Application Types** > **Websphere enterprise applications** > *application_name* > **Map data sources for all 2.x CMP beans** .

This panel displays a table that depicts the EJB modules in your application that contain 2.x CMP beans. Each table row corresponds to a module. A row shows the JNDI name of the data source mapping target

of the EJB module only if you bound them together during application assembly. For every data source that is displayed, you see the corresponding security configuration.

Set Multiple JNDI Names

Specifies the JNDI name to bind to one or more modules. Select one or more modules, click **Set Multiple JNDI Names**, and select the JNDI name for the resource to which you would like to bind the module.

Set Authorization Type

Specifies the authorization type that you to use for the modules. Select one or more modules, click **Set Authorization Type**, and select the authorization type.

You can choose:

- Per application - indicates that the enterprise bean code performs signon.
- Container - indicates that the application server performs signon to the data source.

Modify Resource Authentication Method

Specifies the resource authentication method for the modules that you have configured with container-managed authorization. Select one or more modules, click **Modify Resource Authentication Method**, and select the authentication method.

You can choose between the following authentication methods:

- **None**:
  1. Determine which data source configurations to designate with no authentication method.
  2. Select the appropriate table rows.
  3. Select **None** from the list of authentication method options that precede the table.
  4. Click **Apply**.
- **Use default method (many-to-one mapping)**:
  1. Determine which data source configurations to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. Apply this option to each data source individually if you want to designate different authentication data aliases. See the information center topic on J2EE Connector security for more information on the default mapping configuration.
  2. Select the appropriate table rows.
  3. Select **Use default method (many-to-one mapping)** from the list of authentication method options that precede the table.
  4. Select an authentication data entry or alias from the list.
  5. Click **Apply**.
- **Use Kerberos authentication**: Specifies to use the Kerberos authentication method.
  1. Ensure that you have configured the Kerberos authentication mechanism in the application server.
  2. Select the appropriate table row.
  3. Select **Use Kerberos authentication** from the list of authentication method options that precede the table.
  4. Select an application login configuration from the list.
  5. Click **Apply**.
  6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

  The application server will attempt to verify that you are connecting to the correct type of database when you select this option.
- **Use trusted connections (one-to-one mapping)**:
  1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.

2. Select the appropriate table row.

3. Ensure that the database to which the modules will connect is configured for trusted connections.

4. Select **Use trusted connections (one-to-one mapping)** from the list of authentication method options that precede the table.

5. Select an application login configuration from the list.

6. Click **Apply**.

The application server will attempt to verify that you are connecting to the correct type of database when you select this option.

- **Custom login configuration**:

  1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.

  2. Select the appropriate table row.

  3. Select **Use custom login configuration** from the list of authentication method options that precede the table.

  4. Select an application login configuration from the list.

  5. Click **Apply**.

  6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

Select

Select the check boxes of the rows you want to edit.

EJB Module

Specifies the name of the module that contains the 2.x enterprise beans.

URI

Specifies location of the module relative to the root of the application EAR file.

JNDI name

Specifies the Java Naming and Directory Interface (JNDI) name of the default data source for the EJB module.

Information	Value
Data type	String

Resource authorization

Specifies the authorization type and the authentication method for securing the data source.

Extended Datasource Properties

When selected, you will be directed to a panel on which you can specify extended properties that the module can use for the DB2 data source.

The application server will attempt to verify that you are connecting to the correct type of database when you select this option.

Map data sources for all 2.x CMP beans settings

Use this page to map container-managed persistence (CMP) 2.x beans of an application to data sources that are available to the application.

To view this administrative console page, click **Applications** > **Application Types** > **Websphere enterprise applications** > *application_name* > **Map data sources for all 2.x CMP beans**.

Each table row corresponds to a CMP bean within a specific EJB module. A row shows the JNDI name of the data source mapping target of the bean only if you bound them together during application assembly. For every data source that is displayed, you see the corresponding security configuration.

Set Multiple JNDI names

Specify the Java Naming and Directory Interface (JNDI) name for multiple EJB modules. Select one or more EJB modules from the table, and select a JNDI name from this list to configure the EJB modules with that JNDI name.

Information	Value
Data type	Drop-down list

Set Authorization Type

Specify the authorization type for securing the data source. Select one or more EJB modules from the table to set the authorization type.

Select either **Container** or **Application** from the displayed list. Container-managed authorization indicates that WebSphere Application Server performs signon to the data source. Application-managed authorization indicates that the enterprise bean code performs signon.

Modify Resource Authentication Method

Specify the authorization type and the authentication method for securing the data source. Select one or more EJB modules from the table to modify the resource authentication method.

You can choose between the following authentication methods:

* **None**:
    1. Determine which data source configurations to designate with no authentication method.
    2. Select the appropriate table rows.
    3. Select **None** from the list of authentication method options that precede the table.
    4. Click **Apply**.
* **Use default method (many-to-one mapping)**:
    1. Determine which data source configurations to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. Apply this option to each data source individually if you want to designate different authentication data aliases. See the information center topic on J2EE Connector security for more information on the default mapping configuration.
    2. Select the appropriate table rows.
    3. Select **Use default method (many-to-one mapping)** from the list of authentication method options that precede the table.
    4. Select an authentication data entry or alias from the list.
    5. Click **Apply**.
* **Use Kerberos authentication**: Specifies to use the Kerberos authentication method.
    1. Ensure that you have configured the Kerberos authentication mechanism in the application server.
    2. Select the appropriate table row.
    3. Select **Use Kerberos authentication** from the list of authentication method options that precede the table.
    4. Select an application login configuration from the list.
    5. Click **Apply**.
    6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

    The application server will attempt to verify that you are connecting to the correct type of database when you select this option.

- **Use trusted connections (one-to-one mapping)**:
  1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
  2. Select the appropriate table row.
  3. Ensure that the database to which the modules will connect is configured for trusted connections.
  4. Select **Use trusted connections (one-to-one mapping)** from the list of authentication method options that precede the table.
  5. Select an application login configuration from the list.
  6. Click **Apply**.

  The application server will attempt to verify that you are connecting to the correct type of database when you select this option.
- **Custom login configuration**:
  1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
  2. Select the appropriate table row.
  3. Select **Use custom login configuration** from the list of authentication method options that precede the table.
  4. Select an application login configuration from the list.
  5. Click **Apply**.
  6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

Select
Select the check boxes of the rows that you want to edit.

EJB
The name of an enterprise bean in the application.

EJB Module
The name of the module that contains the enterprise bean.

URI
Specifies location of the module relative to the root of the application EAR file.

Target resource JNDI name
Specifies the resource to which the CMP bean is bound.

Resource authorization
Specifies the current setting for the resource authorization type.

Modify this setting with **Set authorization type**.

Ensure all unprotected 2.x methods have the correct level of protection

Use this page to verify that the unprotected Enterprise JavaBeans (EJB) Version 2.*x* methods have the correct level of protection before you map users to roles.

This administrative console page is displayed during the application deployment process. To access the administrative console page, click **Applications** > **New application** > *application_name*. The page is

displayed as **Ensure all unprotected 2.x methods have the correct level of protection** in the
application deployment steps. On this administrative console page, you can specify whether users can
access specific EJB modules.

To use this administrative console page, select the **Uncheck**, **Exclude**, or **Role** option, the check box next
to the EJB module, and click **Apply**. If you select **Role** option, select the appropriate role for the EJB
module before you click **Apply**.

Uncheck

Select this option if you do not want the application server to verify the access permissions for the EJB
module. Everyone can access the EJB module.

Information	Value
Default:	Selected

Exclude

Select this option to protect this EJB module by making it inaccessible to users regardless of their access
permissions.

Information	Value
Default:	Deselected

Role

Specifies the EJB level of protection based on the security role.

The roles listed in this menu are obtained from the application scope. If the selected role is not in the
module, then it is added to the modules or Java archive (JAR) files.

Information	Value
Default:	Deselected

EJB module

Specifies the name of the module.

If a module name appears in this list, then the module contains unprotected EJB methods.

URI:

Specifies the Uniform Resource Identifier (URI) that is used to locate the Java archive (JAR) file for the
EJB module.

Protection type

Specifies the level of protection that is assigned to a particular module name.

After you select the **Uncheck**, **Exclude**, or **Role** option and click **Apply**, the selected protection option is
displayed in this column.

Provide options to perform the EJB Deploy settings

Use this page 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 page is a step in the application installation and update wizards. To view this page, you must select **Deploy enterprise beans** on the **Select installation options** page. Thus, to view this page, 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 page 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.x 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** page 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 entries, the file names must be fully qualified, separated by a path separator that the target server uses, and enclosed in double quotation marks.

`Windows` On Windows operating systems, the path separator is a semicolon (;). For example:

path\myJar1.jar;*path*\myJar2.jar;*path*\myJar3.jar

`AIX` `HP-UX` `Linux` `Solaris` On the other supported operating systems, the path separator is a colon (:). For example:

path/myJar1.jar:*path*/myJar2.jar:*path*/myJar3.jar

Class path is the same as the **ejbdeploy** command parameter -cp *class_path*.

Information	Value
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*".

Information	Value
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.

Note: 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 on the Provide options to perform the EJB Deploy page, they are deprecated.

Database type is the same as the `ejbdeploy` command parameter `-dbvendor` *name*.

Information	Value
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*`"`.

Information	Value
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`.

Information	Value
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.

Specify the drive and directory where the `sqlj.zip` file resides. For example:

On Windows operating systems, specify `c:\SQLJ\sqlj.zip`.

On all other operating systems, specify `/SQLJ/sqlj.zip`.

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**.

Information	Value
Data type	String
Default	null

JDK compliance level

Specifies the Java developer kit compiler compliance level as *1.4*, *5.0*, *6.0*, or *7.0* when you include application source files for compilation.

The default is to use whatever developer kit version the **ejbdeploy** command is using. For example, if your application is using new functionality defined in Version 7.0 or you are including source files (which is not recommended), then you must specify the Version 7.0 level.

The JDK compliance level that you specify must be the same level as the default Java SDK for the application server to which you are deploying your application. For example, if you select *7.0* for the JDK compliance level on this page, you must ensure that JDK 7.0 is installed and that the default Java SDK for the application server is set to *7.0*. The Java SDKs page of the administrative console lists the software development kits that are installed on the node and enables you to select a default SDK for the node or server. To view the Java SDKs page, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name* > **Java SDKs**.

JDK compliance level is the same as the **ejbdeploy** command parameter `-complianceLevel "1.4"|
"5.0"| "6.0"| "7.0"`.

Information	Value
Data type	String
Default	null (empty string)

Shared library reference and mapping settings

Use the Shared library references and Shared library mapping pages to associate defined shared libraries with an application or web module. A shared library is an external Java archive (JAR) file that is used by one or more applications. Using shared libraries enables multiple applications deployed on a server to use a single library, rather than use multiple copies of the same library. After you associate shared libraries with an application or module, the application or module class loader loads classes represented by the shared libraries and makes those classes available to the application or module.

To view the Shared library references console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Shared library references**. To view the Shared library mapping page, click **Reference shared libraries** on the Shared library references page.

These pages are the same as the Map shared libraries and Map shared libraries to an entire application or module pages in the application installation and update wizards.

On the Shared library references page, the first element listed is the application. The other elements are modules in the application.

To associate shared libraries with your application or module:

1. Select an application or module.
2. Click **Reference shared libraries**.
3. On the Shared library mapping page, select one or more shared libraries that the application or modules uses in the **Available** list, click **>>** to add them to the **Selected** list, and click **OK**.

A defined shared library for a file that your application or module uses must exist to associate your application or module to the library.

If no shared libraries are defined and the application is installed already, on the Shared library mapping page, click **New** and define a shared library.

You can otherwise define a shared library as follows:

1. Click **Environment** > **Shared libraries**.
2. Specify whether the shared library is visible at the cell, node or server level.
3. Click **New**.
4. On the settings page for the new shared library, specify a name and one or more class paths. If the libraries are platform-specific files such as `.dll`, `.so`, or *SRVPGM objects, also specify a native library path. Then, click **Apply**.
5. Save the administrative configuration.

Application

Specifies the name of the application that you are installing or that you selected on the Enterprise applications page.

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.

Shared libraries

Specifies the name of the shared library files associated with the application or module.

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 page of the application installation wizard.

To view this console page 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 to 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 identifers

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 page 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.

Note: If you later change the deployment target of the business-level application or its modules, you must manually update the shared library target to match the target of the application and modules. The targets of shared library composition units are not automatically updated. Not updating the target of the shared library composition unit might cause java.lang.ClassNotFoundException errors and prevent the application or its modules from starting. To prevent these error conditions, also ensure that shared libraries upon which other modules or applications depend have a lower starting weight than dependent applications and modules.

JSP and JSF option settings

Use this page to configure the class reloading of web modules such as JavaServer Pages (JSP) files and to select a JSF implementation to use with this application.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **JSP and JSF options**. This page is the same as the **Provide JSP reloading options for web modules** page on the application installation and update wizards.

The following note applies to the files with an `.xmi` extension in this topic:

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or

module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Web module
Specifies the name of a web module in the installed or deployed application.

URI
Specifies the location of the module that is relative to the root of the application (EAR file).

JSP enable class reloading
Specifies whether to enable class reloading when JSP files are updated.

A web container reloads JSP files only when the IBM extension reloadEnabled in the jspAttributes of the `ibm-web-ext.xmi` file is set to `true`.

Java Platform, Enterprise Edition 5 (Java EE 5) and later applications IBM extension files are in `.xml` file format. For applications versions earlier than Java EE 5, they are in the `.xmi` file format.

JSP reload interval in seconds
Specifies the number of seconds to scan the application file system for updated JSP files. The default is the value of the reloading interval attribute in the IBM extension (`META-INF/ibm-web-ext.xmi`) file of the web module.

To enable reloading, specify a value greater than zero (for example, 1 to 2147483647). The default reload interval is 5. 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.

Java EE 5 applications and later IBM extension files are in `.xml` file format. For applications versions earlier than Java EE 5, they are in the `.xmi` file format.

Sun Reference Implementation 1.2
Select this option to use the Sun Reference Implementation 1.2 JSF implementation.

If you change the JSF implementation that you are using for your application, you must delete any previously compiled JSP files. If you precompiled your application, you must recompile. If you did not precompile, but have already requested JSP files from this application, you must delete the JSP files from the temp directory of your profile.

You can set the JSF engine configuration parameter, com.ibm.ws.jsf.JSF_IMPL_CHECK, to true to automatically mark the JSP files to recompile at application startup.

MyFaces 2.0
Select this option to use the MyFaces JSF implementation. This is the default JSF implementation.

If you change the JSF implementation that you are using for your application, you must delete any previously compiled JSP files. If you precompiled your application, you must recompile. If you did not precompile, but have already requested JSP files from this application, you must delete the JSP files from the temp directory of your profile.

You can set the JSF engine configuration parameter, com.ibm.ws.jsf.JSF_IMPL_CHECK, to true to automatically mark the JSP files to recompile at application startup.

In a mixed-version cell, a Version 7 node uses MyFaces 1.2 if the MyFaces selection is toggled, while a Version 8 and later node uses MyFaces 2.0. For WebSphere Application Server versions before Version 7 (for example, Version 6.1 and earlier), this toggle is ineffective because JSF implementation switching was not supported before Version 7.

Context root for web modules settings

Use this page to specify the context root for web modules during or after installation of an application onto a WebSphere Application Server deployment target.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Context root for web modules**. This page is the same as the Context root for web modules page on the application installation and update wizards.

Web Module

Specifies the name of a web module in the application that you are installing or that you are viewing after installation.

URI

Specifies the location of the module relative to the root of the application EAR file.

Context Root

Specifies the context root of the web application (WAR).

A context root for each web module is defined in the application deployment descriptor during application assembly. Use this field to assign a different context root to a web module. 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`.

Initial parameters for servlets settings

Use this page to specify initial parameters that are passed to the init method of web module servlet filters. You can specify initial parameter values for servlets in web modules during or after installation of an application onto a WebSphere Application Server deployment target. The `<param-value>` values specified in `<init-param>` statements in the `web.xml` file of web modules are used by default.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Init parameters for servlets**. This page is the same as the Init parameters for servlets in each web module panel on the application installation and update wizards.

Module

Specifies the name of a module in the application that you are installing or that you are viewing after installation.

URI

Specifies the location of the module relative to the root of the application (EAR file).

Servlet

Specifies a unique name for the servlet within the application.

A *servlet* is a Java program that uses the Java Servlet Application Programming Interface (API). You must package servlets in a Web archive (WAR) file or web module for deployment to an application server. Servlets run on a Java-enabled web server and extend the capabilities of a web server, similar to the way applets run on a browser and extend the capabilities of a browser.

Name

Specifies the name of the initial parameter passed to the init method of the web module servlet filter.

The following example servlet filter statement in a `web.xml` file specifies an initial parameter name of `attribute`:

```
<init-param>
   <param-name>attribute</param-name>
   <param-value>tests.Filter.DoFilter_Filter.SERVLET_MAPPED</param-value>
</init-param>
```

Value

Specifies the value assigned to an initial parameter passed to the init method of the web module servlet filter.

The following example servlet filter statement in a `web.xml` file specifies an initial parameter value of `tests.Filter.DoFilter_Filter.SERVLET_MAPPED` for the init parameter `attribute`:

```
<init-param>
   <param-name>attribute</param-name>
   <param-value>tests.Filter.DoFilter_Filter.SERVLET_MAPPED</param-value>
</init-param>
```

Description

Specifies information on the initial parameter.

Environment entries for client modules settings

Use this page to configure the environment entries of application client modules that are deployed as Java archive (JAR) files.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Environment entries for client modules**.

This page is the same as the Map environment entries for client modules page on the application installation and update wizards. To view the Map environment entries for client modules page in a wizard, you must select the **Deploy client modules** option on the Select installation options page.

Client module

Specifies the name of a client module.

URI

Specifies the location of the client module relative to the root of the application.

Name

Specifies the name of the environment entry that you are editing or viewing. The environment entry is the `env-entry` property in the client module.

Type

Specifies a data type for the environment entry defined by the `env-entry` property in the client module.

Description
Specifies information about the environment entry.

Value
Specifies an editable value for the environment entry. The value is defined by the `env-entry` property in the client module.

The lookup name is displayed in the **Value** column if the lookup name is configured in the application metadata. The lookup name is not editable. If you do not specify a value on this page, the lookup name is used for the value.

Environment entries for EJB modules settings

Use this page to configure the environment entries of Enterprise JavaBeans (EJB) modules such as entity, session, or message driven beans.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Environment entries for EJB modules**. This page is the same as the Map environment entries for EJB modules page on the application installation and update wizards.

Module
Specifies the name of an EJB module.

URI
Specifies the location of the EJB module relative to the root of the application.

Bean
Specifies the name of an enterprise bean that is contained by the module.

Name
Specifies the name of the environment entry that you are editing or viewing. The environment entry is the `env-entry` property in the EJB module.

Type
Specifies a data type for the environment entry defined by the `env-entry` property in the EJB module.

Description
Specifies information on the environment entry.

Value
Specifies an editable value for the environment entry defined by the `env-entry` property in the EJB module.

The lookup name is displayed in the **Value** column if the lookup name is configured in the application metadata. The lookup name is not editable. If you do not specify a value on this page, the lookup name is used for the value.

Environment entries for web modules settings

Use this page to configure the environment entries of Web modules such as servlets and JavaServer Pages (JSP) files.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Environment entries for web modules**. This page is the same as the Environment entries for web modules page on the application installation and update wizards.

Module

Specifies the name of a web module.

URI

Specifies the location of the module relative to the root of the application (EAR file).

Name

Specifies the name of the environment entry that you are editing or viewing. The environment entry is the `env-entry` property in the web module.

Type

Specifies a data type for the environment entry defined by the `env-entry` property in the web module.

Description

Specifies information on the environment entry.

Value

Specifies an editable value for the environment entry defined by the `env-entry` property in the web module.

The lookup name is displayed in the **Value** column if the lookup name is configured in the application metadata. The lookup name is not editable. If you do not specify a value on this page, the lookup name is used for the value.

Environment entries for application settings

Use this page to configure the environment entries of applications that are deployed as enterprise archive (EAR) files.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Environment entries for the application**.

This page is the same as the Map environment entries for application level page on the application installation and update wizards. To view this page, the application must define one or more environment entries.

Name

Specifies the name of the environment entry that you are editing or viewing. The environment entry is the `env-entry` property in the application.

Type

Specifies a data type for the environment entry defined by the `env-entry` property in the application.

Description

Specifies information about the environment entry.

Value

Specifies an editable value for the environment entry. The value is defined by the `env-entry` property in the application.

The lookup name is displayed in the **Value** column if the lookup name is configured in the application metadata. The lookup name is not editable. If you do not specify a value on this page, the lookup name is used for the value.

Resource environment references

Use this page to designate how the resource environment references of application modules map to remote resources, which are represented in the product as resource environment entries.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Resource environment references**.

Each row of the table depicts a resource environment reference within a specific module of your application. If you bound any references to resource environment entries during application assembly, you see the JNDI names of those resource environment entries in the applicable rows.

To set the mapping relationships between your resource environment references and resource environment entries:

1. Select a row. Be aware that if you check multiple rows on this page, the resource mapping target that you select in step 2 applies to all of those references.
2. Click **Browse** to select a resource environment entry from the new page that is displayed, the Available Resources page. The Available Resources page shows all resource environment entries that are available mapping targets for your application references.
3. Click **Apply**. The console displays the Resource environment references page again. In the rows that you previously selected, you now see the JNDI name of the new resource mapping target.
4. Repeat the previous steps as necessary.
5. Click **OK**. You now return to the general configuration page for your enterprise application.

Table column heading descriptions:

Select
Select the check boxes of the rows that you want to edit.

Module
The name of a module in the application.

EJB
The name of an enterprise bean that is accessed by the module.

URI
Specifies location of the module relative to the root of the application EAR file.

Reference binding
The name of a resource environment reference that is declared in the deployment descriptor of the application module. The reference corresponds to a resource that is bound as a resource environment entry into the JNDI name space of the application server.

JNDI name
The Java Naming and Directory Interface (JNDI) name of the resource environment entry that is the mapping target of the resource environment reference.

Information	Value
Data type	String

Message destination reference settings

If your application uses message-driven beans, use this page to specify the Java Naming and Directory Interface (JNDI) name of the J2C administered object to bind the message destination reference to the message-driven beans. You must map each message destination reference that is defined in your application to an administered object.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Message destination references**. This page is the same as the **Bind message destination references to administered objects** page on the application installation and update wizards.

If the message destination reference is from an EJB 3.0 or later module, then the JNDI name is optional and the run time provides a container default value.

Attention: 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.

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. Complete one of the following steps:
   - Specify a message destination name.
   - Select a target resource JNDI name for a message destination.
4. Click **Apply**.
5. Click **OK** or **Next**.

Module
Specifies the name of the module that contains the bean.

Bean
Specifies name of a bean in the application.

URI
Specifies the location of the module relative to the root of the enterprise archive (EAR) file.

Message destination object
Specifies the message destination object.

Type
Specifies the type of object.

Target Resource JNDI Name
Specifies the Java Naming and Directory Interface (JNDI) name of the bean.

This is a data entry field. To change the JNDI name bound to this bean, type the new name in this field.

Select current backend ID settings

Use this page to select a backend identifier for container-managed persistence (CMP) beans that contain mappings for multiple backend databases.

This administrative console page is a step in the application installation and update wizards. To view this administrative console page, click **Applications** > **New Application** > **New Enterprise Application** > *application_path* > **Next** > **Detailed - Show all installation options and parameters** > **Next** > **Next** or **Continue** > **Step: Select current backend ID**. This page is displayed in the wizards if **Database type** is blank on the Provide options to perform the EJB Deploy page.

A backend can represent different database vendors, or simply alternative mappings and table qualifiers. If a Java archive (JAR) file for an enterprise bean defines CMP beans that contain mappings for multiple backend databases, you must select a current backend ID to be used when the module is installed on a deployment target. The backend ID determines the persister classes that get loaded at deployment.

Module
Specifies the name of the module that contains the bean.

URI
Specifies the location of the module relative to the root of the application EAR file.

Current backend ID
Specifies the current backend ID to be used when the module is installed on a deployment target.

Provide JNDI names for JCA objects settings

Use this page to configure Java Naming and Directory Interface (JNDI) name values for J2C objects (J2CConnectionFactory, J2CActivationSpec, and J2CAdminObject) in your application or modules. If your application contains an embedded resource archive (RAR) file, specify the name and JNDI name of each JCA connection factory, administered object, and activation specification.

This administrative console page is a step in the application installation and update wizards. To view this administrative console page, click **Applications** > **New Application** > **New Enterprise Application** > *application_path* > **Next** > **Detailed - Show all installation options and parameters** > **Next** > **Next** or **Continue** > **Step: Provide JNDI names for JCA objects**.

Connector module
Specifies the name of a connector module of the RAR file.

URI
Specifies the location of the module that is relative to the root of the RAR file.

Object identifier
Specifies the name of the J2C object. The object can be a JCA connection factory, administered object, or activation specification.

Bindings
Specifies the name and Java Naming and Directory Interface (JNDI) name of the J2C object.

These are data entry fields. To change the name or JNDI name bound to this object, type the new names in the fields.

Correct use of the system identity

Use this page to manage the system identity properties for the Enterprise JavaBeans (EJB) method in your application.

This administrative console page is displayed during the application deployment process. To access the administrative console , click **Application** > **New application** > **New Enterprise Application**. The is displayed as *Correct use of System Identity* in the application deployment steps.

To use this page, complete the following steps:
1. Select an application that supports security and click **Next**.
2. Select **Detailed - Show all installation options and parameters** and click **Next**.
3. Select the **Correct use of system identity** step.

Bean

A component that implements a business task or business entity and resides in an EJB container. Entity beans, session beans, and message-driven beans are all enterprise beans.

Module

In Java EE programming, a software unit that consists of one or more components of the same container type and one deployment descriptor of that type. Examples include EJB, Web, and application client modules.

URI

A Uniform Resource Identifier (URI) is a unique address that is used to identify content on the Web, such as a page of text, a video or sound clip, a still or animated image, or a program.

Method signature

The combination of a name of a method along with the number and types of the parameters and their order.

Role

Specifies the RunAs role that is used for this EJB method.

Username

Specifies the user name that is assigned to the RunAs role for this EJB method.

The user name is used in conjunction with the RunAs role that you select for the Role.

Requirements for setting data access isolation levels

This article discusses the criteria and effects of setting isolation levels for data access components that comprise Enterprise JavaBeans (EJB) 2.x and later modules.

In an EJB 1.1 module, you can set the isolation level at the method level or bean level. This capability also applies to container-managed persistence (CMP) 1.1 beans that you assemble into *EJB 2.x modules*. WebSphere Application Server permits the deployment descriptor of a CMP bean to declare the version level of 1.1, regardless of the overall module version.

However, the ability to set isolation level at the method or bean level does **not** apply to other enterprise beans within an EJB 2.x module, including *CMP 2.x beans*. WebSphere Application Server Version 5.0 removed this capability from EJB 2.0 modules to deliver an architecture that ultimately provides more efficient connection use.

Consequently, later versions of the product enforce the following restrictions on declaring isolation level for CMP 2.x beans—as well as session beans, message-driven beans, and bean managed persistence (BMP) beans that you assemble into EJB 2.x modules:

- You cannot specify isolation level on the EJB method level or bean level.
- If you configure a JDBC application, a bean-managed persistence (BMP) bean, or a servlet to participate in global transactions, any connection that is shared cannot accept a user-specified isolation level. WebSphere Application Server can only set a user-specified isolation level on a connection that is not shared within a global transaction. *Generally, you want to refrain from specifying isolation levels on shareable connections.*

The configuration for the isolation level is determined by the type of bean that is used by the component:

Isolation level on connections used by 2.x CMP beans
> In a EJB 2.x module, when a CMP 2.x bean uses a new data source to access a backend database, the isolation level is determined by the WebSphere Application Server run time, based on the type of access intent assigned to the bean or the calling method. Other non-CMP

connection users can access this same data source and also use the access intent and application profile support to manage their concurrency control.

Connections used by other 2.x enterprise beans and other non-CMP components

For all other JDBC connection instances (connections other than those used by CMP beans), you can specify an isolation level on the data source resource reference. For shareable connections that run in global transactions, this method is the only way to set the *isolationLevel* for connections. Trying to directly set the isolation level through the *setTransactionIsolation()* method on a shareable connection that runs in a global transaction is not allowed. To use a different isolation level on connections, you must provide a different resource reference. Set these defaults through your assembly tool.

Each resource reference associates with one isolation level. When your application uses this resource reference Java Naming and Directory Interface (JNDI) name to look up a data source, every connection returned from this data source using this resource reference has the same isolation level.

Components needing to use shareable connections with multiple isolation levels can create multiple resource references, giving them different JNDI names, and have their code look up the appropriate data source for the isolation level they need. In this way, you use separate connections with the different isolation levels enabled on them.

It is possible to map these multiple resource references to the same configured data source. The connections still come from the same underlying pool, however; the connection manager does not allow sharing of connections requested by resource references with different isolation levels. Consider the following scenario:

- A data source is bound to two resource references: *jdbc/RRResRef* and *jdbc/RCResRef*.
- RRResRef has the *RepeatableRead* isolation level defined. RCResRef has the *ReadCommitted* isolation level defined.

If your application wants to update the tables or a BMP bean updates some attributes, it can use the *jdbc/RRResRef* JNDI name to look up the data source instance. All connections returned from the data source instance have a RepeatableRead isolation level. If the application wants to perform a query for read only, then it is better to use the *jdbc/RCResRef* JNDI name to look up the data source.

The product does not require you to set the isolation level on a data source resource reference for a non-CMP application module. If you do not specify isolation level on the resource reference, or if you specify TRANSACTION_NONE, the WebSphere Application Server run time uses a default isolation level for the data source. Application Server uses a default setting based on the JDBC driver.

For most drivers, WebSphere Application Server uses an isolation level default of TRANSACTION_REPEATABLE_READ. For Oracle drivers, however, Application Server uses an isolation level of TRANSACTION_READ_COMMITTED. Use the following table for quick reference:

Database:	Default isolation level:
DB2	RR
Oracle	RC
Sybase	RR
Informix	RR
Apache Derby	RR
SQL Server	RR

Note: These same default isolation levels are used in cases of direct JNDI lookups of a data source.

- RR = JDBC Repeatable read (TRANSACTION_REPEATABLE_READ)
- RC = JDBC Read committed (TRANSACTION_READ_COMMITTED)

To customize the default isolation level, you can use the webSphereDefaultIsolationLevel custom property for the data source. In most cases you should define the isolation level in the deployment descriptor when you package the EAR file, but in certain situations you might need to customize the default isolation level. This property will have no effect if any of the previous options are used, and this custom property is provided for those situations in which there is no other means of setting the isolation level.

Use the following values for webSphereDefaultIsolationLevel custom property:

Possible values	JDBC isolation level	DB2 isolation level
8	TRANSACTION_SERIALIZABLE	Repeatable Read (RR)
4 (default)	TRANSACTION_REPEATABLE_READ	Read Stability (RS)
2	TRANSACTION_READ_COMMITTED	Cursor Stability (CS)
1	TRANSACTION_READ_UNCOMMITTED	Uncommitted Read (UR)

To define this custom property for a data source:

1. Click **Resources** > **JDBC provider** > **JDBC_provider**.
2. Click **Data sources** in the Additional Properties section.
3. Click the name of the data source.
4. Click **Custom properties**.
5. Create the webSphereDefaultIsolationLevel custom property.
   a. Click **New**.
   b. Enter webSphereDefaultIsolationLevel for the name field.
   c. Enter one of the possible values in the value field.

Application Server sets the isolation level by prioritizing the available settings. Application Server will set the isolation level based on the values for the following, in this order:

1. Resource reference isolation level
2. Isolation level that is specified by the access intent policy
3. Custom property that configures an isolation level
4. Application Server's default setting.

Metadata for module settings

Use this page to instruct a Java Platform, Enterprise Edition (Java EE) enterprise bean (EJB) deployment descriptor, web module deployment descriptor, or JCA resource adapter archive (RAR) module to ignore annotations that specify deployment information.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > **application_name** > **Metadata for modules**. This page is the same as the Metadata for modules page on the application installation and update wizards.

If your application contains Java EE 5 or later modules, you can select to lock the deployment descriptor of one or more of the modules on the Metadata for modules page. If you select a **metadata-complete attribute** check box (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 or later 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. If you select to lock deployment descriptors, the product merges Java EE annotation-based metadata with the XML-based existing deployment descriptor metadata and persists the result.

When applications contain a large number of Java classes, the deployment processing time for the annotations can increase. To minimize the performance impact, you can use one of the following methods:

- Determine whether the module needs to use Java EE 5 or 6. If the module does not need to use Java EE 5 or 6, the annotations within the Java classes are not scanned.
- Use the "metadata-complete attribute" in the module descriptor if the module uses Java EE 5 or later and it does not contain any annotations. This attribute disables the annotations processing for the module, but Java EE 5 or later modules might still be placed in the descriptor file. If you are migrating your application, but you are not adding annotations, consider using this attribute value.
- Restructure the application to place the utility Java archive (JAR) files into shared libraries if those JAR files do not have annotation information. Consider this method if you cannot set the "metadata-complete attribute."
- Move the JAR files in the `WEB-INF/lib` directory to the root directory of the enterprise archive (EAR) file. Nested archives, such as a JAR file that is within a web application archive (WAR) that is within an EAR file, are very cumbersome to search through because of the multiple levels of compression.

Module

Specifies the name of a module in the installed (or deployed) application.

Information	Value
Data type	String

URI

Specifies the location of the module relative to the root of the EAR file.

Information	Value
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.

By default, a **metadata-complete attribute** check box is not selected and the product does not write out annotation data to a module deployment descriptor.

If your modules do 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 annotation data to a module deployment descriptor.

Note: If your Java EE 5 or later application uses annotations and a shared library, do not select **metadata-complete attribute**. When your application uses annotations and a shared library, setting the `metadata-complete` attribute to `true` causes the product to incorrectly represent an `@EJB` annotation in the deployment descriptor as <ejb-ref> rather than <ejb-local-ref>. For web modules, setting the `metadata-complete` attribute to `true` might cause InjectionException errors. If you must select **metadata-complete attribute** (set the `metadata-complete` attribute to `true`), avoid errors by not using a shared library, by placing the shared library in either the `classes` or `lib` directory of the application server, or by fully specifying the metadata in the deployment descriptors.

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 the check boxes, the link to this page is no longer shown on the enterprise application settings page.

Information	Value
Data type	Boolean
Default	false (deselected)

Provide options to perform the web services deployment settings

Use this page to specify options for web services deployment.

This administrative console page is a step in the application installation and update wizards.

To view this page, you must select **Deploy web services** on the **Select installation options** page.

To view this administrative console page, complete the following steps:

1. Click **Applications** > **New application** > *application_path* .
2. Select the option to **Show all installation options and parameters** .
3. Click **Next** to get to the **Step: Select installation options** page.
4. Select **Deploy web service**.
5. Click **Next** to get to the **Step: Provide options to perform the web services deployment** page.

You can specify the web services deployment options on this page only when installing or updating an application that uses web services.

The `wsdeploy` command is supported by Java API for XML-based RPC (JAX-RPC) applications. The Java API for XML-Based Web Services (JAX-WS) programming model that is implemented by the application server does not support the `wsdeploy` command. If your web services application contains only JAX-WS endpoints, you do not need to run the `wsdeploy` command, as this command is used to process only JAX-RPC endpoints.

The options that you specify set parameter values for the `wsdeploy` command. The `wsdeploy` command adds product-specific deployment classes to a web services-compatible enterprise archive (EAR) file or an application client Java archive (JAR) file. These classes include:
* Stubs
* Serializers and deserializers
* Implementations of service interfaces

The `wsdeploy` command is run during installation after you click **Finish** on the **Summary** page of the wizard.

Deploy web services option - Classpath

Specifies entries to add to the CLASSPATH when the generated classes are compiled.

To specify the class paths of multiple entries, you need to separate the entries with a semicolon on Windows platforms and on Linux, Unix, and z/OS platforms, you need to use a colon to separate the entries. This is the same separator that is used with the CLASSPATH environment variable.

This option is the same as the `wsdeploy` command parameter `-cp` *class_path*.

Information	Value
Data type	String
Default	null

Deploy web services option - Extension Directories

Specifies a directory that contains zipped or Java archive (JAR) files. All zipped and JAR files in this directory are added to the CLASSPATH used to compile the generated files.

This option is the same as the **wsdeploy** command parameter `-jardir directory`.

Information	Value
Data type	String
Default	null

Display module build ID settings

Use this page to view the build identifier of a module in a Java Platform, Enterprise Edition (Java EE) enterprise archive (EAR file). The build identifier for a module is shown if the `MANIFEST.MF` file of a module or application specifies a build identifer.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Display module build IDs**. This page is the same as the Display module build IDs page on the application installation and update wizards.

Module

Specifies the name of a module in the installed (or deployed) application.

Information	Value
Data type	String

URI

Specifies the location of the module relative to the root of the application EAR file.

Information	Value
Data type	String

Build ID

Specifies the build identifier for a module if the `MANIFEST.MF` file specifies a build identifer.

You cannot modify the build ID on this page because this field is read-only.

Information	Value
Data type	String

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

Note: Application installation using the Java EE Application Deployment API specification (JSR-88) was deprecated in WebSphere Application Server Version 8.0. Use another option to deploy applications to a server. The closest option to using the Java EE Deployment API is using Java Management Extensions (JMX) MBean programming. For information on deployment options, see "Ways to install enterprise applications or modules."

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.

Procedure

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.

Table 31. JAR files that contain the manifest attribute.   Enable your code to find the DeploymentFactory using the JAR manifest attribute.

Environment	JAR file containing the manifest attribute
Application server	*app_server_root*/plugins/com.ibm.ws.admin.services.jar
Application client	*app_client_root*/plugins/com.ibm.ws.j2ee.client.jar
Thin application client	*app_client_root*/runtimes/com.ibm.ws.admin.client_8.0.0.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 URI, 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 URI for deployment has the following 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.

   You can specify an Internet Protocol Version 6 (IPv6) address for the *host* element in the URI for deployment. Enclose the IPv6 address in square brackets ([]); for example:

   `"deployer:WebSphere:[IPv6_address]:port"`

   Also, you can add an optional parameter, `connectorType`, to the URI for deployment. For example, to use the RMI connector to access `myserver`, code the URI 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 stand-alone modules during deployment using the DConfigBean class in the Java EE Application Deployment API specification (JSR-88).

Before you begin

Note: Application installation using the Java EE Application Deployment API specification (JSR-88) was deprecated in WebSphere Application Server Version 8.0. Use another option to deploy applications to a server. The closest option to using the Java EE Deployment API is using Java Management Extensions (JMX) MBean programming. For information on deployment options, see "Ways to install enterprise applications or modules."

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 J2EE 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.Map objects. The map entries contain configuration attributes, for which your code can get and set values.

Procedure

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 map.
      int index = 0;
      Map map = (Map)
         ipDesc.getIndexedReadMethod().invoke
            (dcBean, new Object[]{new Integer(index)});

      while (map != null)
      {
         // Iterate over the map and set values for attributes.

         // Set the map back into the DCBean.
         ipDesc.getIndexedWriteMethod().invoke
            (dcBean, new Object[]{new Integer(index), map});

         // Get the next entry in the indexed property
         map = (Map)
            ipDesc.getIndexedReadMethod().invoke
               (dcBean, new Object[]{new Integer(++index)});
      }
   }
   ```

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 and deployment descriptors of 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.

Attention: 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.

Note:

- You must use either the administrative console or wsadmin scripting to synchronization a node. Of these two options, using the administrative console is the best way to perform this operation. The Nodes panel in the administrative console includes the **Synchronize** operation.

  If you need to use wsadmin scripting to synchronize a node, use the NodeSync mbean's sync() command.

- Do not restart the node agent as part of the synchronize node process. Administration operations, such as node synchronization for application deployment, or updates that take place while the node agent is starting, that are initiated through the node agent, and affect the application servers, fail until the node agent has a chance to discover the application servers.

Complete the following procedure to change the settings of an application or module using the administrative console.

Procedure

- 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.
- 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 enterprise 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.

If you updated module metadata while the application was running, restarting the application might not be sufficient for the changes to take effect. For example, if you changed descriptors in running Java EE 6 applications that use annotations, you must reinstall the application. If you changed classes that introduce, remove, or alter class hierarchies within an application, and those changes impact annotated classes, you also must reinstall the application.

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 or later binding definitions support files with the suffix of XML for EJB 3.x and Web 2.5 and later 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`.

This topic provides the following information about bindings:
- "Times when bindings can be defined"
- "Required bindings" on page 349
- "Application resource conflicts" on page 353

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.x and Web 2.5 and later 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.x or Web 2.5 and later 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 or later 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.

  Restriction: 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.xmi` 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 page. 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.x home or business interfaces on your enterprise beans because the EJB container assigns default bindings.

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.xmi`. A deployer can also specify whether authentication is handled at the container or application level.

WebSphere Application Server Version 8.x supports CMP beans in EJB 2.x or 1.x modules. Version 8.x does not support CMP beans in EJB 3.0 modules.

If a deployer chooses to generate default bindings when installing the application, the installation 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 page and on the 2.x entity bean data sources page. 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 page and on the Provide default data source mapping for modules containing 1.x entity beans page. After installation, access console pages like 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 page. Selecting **Deploy enterprise beans** enables you to access the Provide options to perform the EJB Deploy page.
2. On the Provide options to perform the EJB Deploy page, set the database type to "" (null).

During application installation, if you select **Deploy enterprise beans** on the Select installation options page and specify a database type for the EJB deployment tool on the Provide options to perform the EJB Deploy page, 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 or later 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, or in another `java:` namespace if the reference name is a `java:module`, `java:app`, or `java:app` URL. EJB references with URL names are bound into the corresponding namespace according to the URL.

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 installation 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 page. 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 page or select **Allow EJB reference targets to resolve automatically** on the Select installation options, Map EJB references to beans, or EJB references console pages.

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 that do not use a URL for the JNDI name are made available as follows:

Table 32. Resource reference subcontexts. JNDI `java:comp/env` names are used for resource reference subcontexts.

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`

Applications alternatively can assign names to resource references that are `java:` URLs with prefixes such as `java:module`, `java:app`, and `java:global`. The URLs map to namespaces other than the component namespace, which contains `java:comp/env` name bindings. Resource references with URL names are bound into the corresponding namespace according to the URL.

For each resource reference, you must specify a JNDI name. If a deployer chooses to generate default bindings when installing the application, the installation 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 page. 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 page.

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.xmi`.

If a deployer chooses to generate default bindings when installing the application, the installation 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 page, 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.xmi`.

If a deployer chooses to generate default bindings when installing the application, the installation wizard assigns JNDI names to incomplete bindings.

- For EJB 2.0 or later message-driven beans deployed as JCA 1.5-compliant resources, the installation 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 page. 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 page.

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**.

> **Restriction:** You can only bind message driven-beans that are defined in an EJB 3.0 or later module to an activation specification.

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 installation 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`.

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.

Application resource conflicts

Before Version 8 of the product, application-defined resources such as references and environment entries were bound into the component namespace relative to `java:comp/env`. In Version 8.0 and later, an application developer can assign a name to a resource that is a `java:` URL prefixed with `java:module`, `java:app`, or `java:global`. Each of these URLs resolves to distinct namespaces, different from the component namespace. A `java:module` namespace is shared among all components in a module, a `java:app` namespace is shared among all modules in an application, and a `java:global` namespace is shared among all applications in a cell. Because the namespaces are shared, different resources might be assigned the same name, resulting in conflicts.

Conflicts at the module scope can occur only if two components in the module define resources with the same name. Because of the small size of this scope, it is unlikely for a module to have true conflicts. However, if multiple instances of the same resource definition exist, they must be configured the same. For example, two EJB references to a particular EJB type which are both assigned the same `java:module` name must both be configured with the same binding data. Otherwise, the two resources will conflict and the application configuration action will fail.

Application-scoped resources are like module-scoped resources. The only difference is that the definitions can originate from any module in the application. As with module-scoped resources, all application-scoped resources that have the same name must be the same type of resource and must be configured with the same binding data.

Global-scoped resources differ from application- and module-scoped resources in that conflicts can occur among different applications. When a conflict occurs, conflicting applications cannot coexist if the resources are not logically the same. If multiple occurrences of a global-scoped resource all identify logically the same resource, they must all be configured with the same binding data in order to not be detected by the product as conflicting. To edit a global-scoped resource for which occurrences for multiple applications exist, all defining applications must be edited in the same session so as not to introduce a conflict. Failure to do so will result in a failure when the session is saved.

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:

Table 33. Button descriptions.   Use the buttons to manage enterprise applications.

Button	Resulting action
Start	Attempts to run the application. After the application starts up successfully, the state of the application changes to `Started` if the application starts up on all deployment targets, else the state changes to `Partial Start`.
Stop	Attempts to stop the processing of the application. After the application stops successfully, the state of the application changes to `Stopped` if the application stops on all deployment targets, else the state changes to `Partial Stop`.
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.

Table 33. Button descriptions (continued). Use the buttons to manage enterprise applications.

Button	Resulting action
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. **gotcha:** You can use the com.ibm.websphere.management.application.client.EnvEntry_processBindings JVM custom property to specify how an environment entry value is handled during an application deployment. However: • If an application is deployed with com.ibm.websphere.management.application.client.EnvEntry_processBindings enabled and is then exported to be deployed on a pre-Version 8.0 environment, the environment entry value in the XML bindings are not recognized by both the deployment and runtime environments. • If an application is deployed with com.ibm.websphere.management.application.client.EnvEntry_processBindings enabled and is then exported to be deployed on a Version 8.0 or 8.0.0.1 environment, the environment entry value in XML bindings are not be recognized by the deployment environment but are recognized by the runtime environment.

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.

Table 34. Application status. The status indicates whether the application is running.

Icon	Application status	Description
	Started	Application is running.
	Partial Start	Application is in the process of changing from a `Stopped` state to a `Started` state. Application is starting to run but is not fully running yet. Or, it cannot fully start because 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 `Started` state to a `Stopped` 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**.

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*.

Attention: 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:

Table 35. Characters that you cannot use in a name. The product does not support these characters in a name.

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	

Information	Value
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.

Information	Value
Data type	String
Default	Issue warnings

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.

Attention: 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.

Procedure

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.

   best-practices: 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 stand-alone 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**.

Attention: 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: Be aware of the following limitations:

- The first application might not start completely before the application server tries to start the next application in the startup order. This scenario might cause unexpected results if the second application depends on the first application.

- 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.

Information	Value
Data type	Integer

Information	Value
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.

Information	Value
Data type	Boolean
Default	false

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.

Information	Value
Data type	Boolean
Default	true

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.

Attention: 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.

Procedure

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:

C:/WebSphere/AppServer/profiles/*profile_name*/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.

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.

   Important: 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**.

Attention: 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.

  Important: 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.

Information	Value
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.

Information	Value
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.

Important: 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.

Information	Value
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.

Table 36. File permission string sets for list options.   Select a list option or specify a file permission string in the text field.

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.

best-practices: 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:

Table 37. Example URIs for file permission operations.   Results are shown following this table.

Number	Example URI
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

best-practices: Regardless of the operation system, always use a forward slash (/) as a file path separator in file patterns.

> Windows   You cannot unset read permission on a file on Windows platforms. With POSIX style permission bits, the bit for denoting `readable` on a file is 4, `writable` is 2, and `executable` is 1. Thus, permission of a file on a Windows platform is either 5 or 7. Also, in POSIX style there are `user`, `group` and `world` permissions. You can only set the `user` permission for a file on Windows platforms. The `group` and `world` permission bits are ignored.

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.

Information	Value
Data type	String

Application build level:

Specifies an uneditable string that identifies the build version of the application.

Information	Value
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.

The following note applies to the xmi file references in this topic:

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.

- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

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 JavaBeans (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 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.

Attention: 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.

Procedure

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Class loading and update detection** to access the Class loading and update detection page.

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 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.
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 `Classes loaded with local class loader first (parent last)`, your application can override classes contained in the parent class loader. **Attention:** Specifying the `Classes loaded with local class loader first (parent last)` value might result in LinkageErrors or ClassCastException messages if you have mixed use of overridden classes and non-overridden classes.

5. 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.

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.

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**.

Attention: 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.

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

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. The following table shows the reload settings in the `deployment.xml` file based on the setting of reloadEnabled.

Table 38. JSP and servlet class reloading. The following table shows the reload settings in the `deployment.xml` file based on the setting of reloadEnabled.

deployment.xml	ibm-web-ext.xmi	ibm-web-ext.xmi (reloadingEnabled)	ibm-web-ext.xmi (jspAttributes name="reloadEnabled"	Change in JSP	Change in servlet class
reloadEnabled = false	reloadInterval set or not	true	true	Translate, recompile, reload	loading new class
reloadEnalbed = false		false	true	Translate, recompile, reload	NO loading of new class
reloadEnalbed = false		true	false	NO translate, recompile, reload	Loading new class
reloadEnalbed = true	reloadInterval > 0	false or true	false	NO translation or reload of JSP	Loading new class at specified interval
reloadEnabled = true		false or true	true	Translation and reload JSP	Loading new class
reloadEnabled = true	reloadInterval = 0	false or true	false	NO translation or reload of JSP	NO loading of new class
reloadEnabled = true	reloadInterval = 0	false or true	true	Translation and reload of JSP	NO loading of new class

- reloadEnabled = `false` in the `deployment.xml` file means that the properties in the `ibm-web-ext.xmi` file will be in effect.(whatever is set for reloadingEnabled in the `ibm-web-ext.xmi` file.
- reloadEnabled = `true` in the `deployment.xml` AND reloadInterval = value of polling interval set to a value greater than 0 by using the administrative console, causes reloading in time interval specified by reloadInterval.
- reloadEnabled = `true` in the `deployment.xml` AND reloadInterval= 0 causes no class reloading.

Note:

- If you want to ensure that the reloading of any servlet class is disabled no matter what is set in the `ibm-web-ext.xmi` file, then make sure the reloadInterval in the `ibm-web.ext.xmi` file is set to 0.
- If jspAttributes name="reloadEnabled" is NOT present in the `ibm-web-ext.xmi` file, then the reloadingEnabled value becomes the operative reload action.

Information	Value
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.

Information	Value
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.

Information	Value
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.

Information	Value
Data type	String
Default	Class loader for each WAR file in application

Manage modules settings

Use this page 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 page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Manage modules**. This page is the similar to the Map modules to servers page on the application installation and update wizards.

On this page, 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 8.x features on a Version 7.x or 6.x target server. Similarly, you cannot install modules that use Version 7.x features on a Version 6.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**.

Attention: 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 page from a console enterprise application page for an already installed application, you can also use this page 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:

Table 39. Button descriptions. *Use the buttons to manage modules.*

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.

Table 39. Button descriptions (continued).   Use the buttons to manage modules.

Button	Resulting action
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) archive (SAR), web module, or client module on a Version 8.x target under any of the following conditions:
• The module supports Java Platform, Enterprise Edition (Java EE) 6.
• The module calls an 8.x runtime application programming interface (API).
• The module uses an 8.x product feature.

You must install an application, EJB, SAR, or web module on a Version 8.x or 7.x target under any of the following conditions:
• The module supports Java EE 5.
• The module calls a 7.x runtime 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, 7.x or 8.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, 7.x or 8.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 or 8.x deployment target.

You can install an application or module developed for a Version 5.x product on any 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 page 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 8.x features cannot be installed onto Version 7.x or 6.x target servers. Similarly, modules that use Version 7.x features cannot be installed onto Version 6.x target servers.

About this task

During application installation, different deployment targets might have been specified.

You use the Manage modules 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 enterprise application settings page.

On the Manage modules page, 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.

Attention: 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.

Procedure

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 8.x features on a Version 7.x or 6.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 or 6 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 8 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 topic 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 enterprise application installation using the console and, after the application is installed, can be accessed from an enterprise application settings page.

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.

Procedure

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 stand-alone 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 page 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 page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Virtual hosts**. This page is the same as the **Map virtual hosts for web modules** page on the application installation and update wizards.

On this page, 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**.

Attention: 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 or later applications and 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 or later annotations.

Unless an application supports Java EE 5 or later, 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.

Procedure

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:

Table 40. Accessing View deployment descriptor pages in the console.   Click links on the console navigation tree and pages.

Module	Deployment descriptor file	Accessing a console view
Enterprise application	`application.xml`	**Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **View deployment descriptor**
Web application	`WEB-INF/web.xml`	**Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Manage modules** > *module_name* > **View deployment descriptor**
	`WEB-INF/portlet.xml`	**Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Manage modules** > *module_name* > **View portlet deployment descriptor**
Enterprise bean	`ejb-jar.xml`	**Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Manage modules** > *module_name* > **View deployment descriptor**
Application client	`application-client.xml`	**Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Manage modules** > *module_name* > **View deployment descriptor**
Web service	`webservices.xml`	**Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Manage modules** > *module_name* > • **View web services client deployment descriptor extension** • **View web services server deployment descriptor** • **View web services server deployment descriptor extension** For information about the views, see the topic on viewing Web services deployment descriptors in the administrative console.
Resource adapter embedded in enterprise application	`ra.xml`	**Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Manage modules** > *ra_module_name* > **View deployment descriptor**
Stand-alone Resource adapter	`ra.xml`	**Resources** > **Resource Adapters** > **Resource adapters** > *module_name* > **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 application 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 or later 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 Java EE 5 or later 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**.

Note: If your Java EE 5 or later application uses annotations and a shared library, do not select **metadata-complete attribute**. When your application uses annotations and a shared library, setting the `metadata-complete` attribute to `true` causes the product to incorrectly represent an `@EJB` annotation in the deployment descriptor as `<ejb-ref>` rather than `<ejb-local-ref>`. For web modules, setting the `metadata-complete` attribute to `true` might cause InjectionException errors. If you must set the `metadata-complete` attribute to `true`, avoid errors by not using a shared library, by placing the shared library in either the `classes` or `lib` directory of the application server, or by fully specifying the metadata in the deployment descriptors.

Metadata for module settings

Use this page to instruct a Java Platform, Enterprise Edition (Java EE) enterprise bean (EJB) deployment descriptor, web module deployment descriptor, or JCA resource adapter archive (RAR) module to ignore annotations that specify deployment information.

To view this administrative console page, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Metadata for modules**. This page is the same as the Metadata for modules page on the application installation and update wizards.

If your application contains Java EE 5 or later modules, you can select to lock the deployment descriptor of one or more of the modules on the Metadata for modules page. If you select a **metadata-complete attribute** check box (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 or later 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. If you select to lock deployment descriptors, the product merges Java EE annotation-based metadata with the XML-based existing deployment descriptor metadata and persists the result.

When applications contain a large number of Java classes, the deployment processing time for the annotations can increase. To minimize the performance impact, you can use one of the following methods:

- Determine whether the module needs to use Java EE 5 or 6. If the module does not need to use Java EE 5 or 6, the annotations within the Java classes are not scanned.
- Use the "metadata-complete attribute" on page 341 in the module descriptor if the module uses Java EE 5 or later and it does not contain any annotations. This attribute disables the annotations processing for the module, but Java EE 5 or later modules might still be placed in the descriptor file. If you are migrating your application, but you are not adding annotations, consider using this attribute value.
- Restructure the application to place the utility Java archive (JAR) files into shared libraries if those JAR files do not have annotation information. Consider this method if you cannot set the "metadata-complete attribute" on page 341.
- Move the JAR files in the `WEB-INF/lib` directory to the root directory of the enterprise archive (EAR) file. Nested archives, such as a JAR file that is within a web application archive (WAR) that is within an EAR file, are very cumbersome to search through because of the multiple levels of compression.

Module
Specifies the name of a module in the installed (or deployed) application.

Information	Value
Data type	String

URI
Specifies the location of the module relative to the root of the EAR file.

Information	Value
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.

By default, a **metadata-complete attribute** check box is not selected and the product does not write out annotation data to a module deployment descriptor.

If your modules do 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 annotation data to a module deployment descriptor.

Note: If your Java EE 5 or later application uses annotations and a shared library, do not select **metadata-complete attribute**. When your application uses annotations and a shared library, setting the `metadata-complete` attribute to `true` causes the product to incorrectly represent an `@EJB` annotation in the deployment descriptor as `<ejb-ref>` rather than `<ejb-local-ref>`. For web modules, setting the `metadata-complete` attribute to `true` might cause InjectionException errors. If you must select **metadata-complete attribute** (set the `metadata-complete` attribute to `true`), avoid errors by not using a shared library, by placing the shared library in either the `classes` or `lib` directory of the application server, or by fully specifying the metadata in the deployment descriptors.

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 the check boxes, the link to this page is no longer shown on the enterprise application settings page.

Information	Value
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.

Procedure

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 `Started`. The status is changed to `partially started` 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 *Stopped*.

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.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

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 topic.

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. 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.

Procedure

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 8.x, 7.x or 6.x deployment target.

A *deployment target* is a server with all members on a WebSphere Application Server product. For example, an *8.5 deployment target* is a server with all members on a WebSphere Application Server Version 8.5 product.

An application, enterprise bean (EJB) module, Session Initiation Protocol (SIP) archive (SAR), web module, or client module must be installed on a Version 8.x target under any of the following conditions:
- The module supports Java Platform, Enterprise Edition (Java EE) 6.
- The module calls an 8.x runtime application programming interface (API).
- The module uses an 8.x product feature.

An application, EJB, SAR, or web module must be installed on a Version 8.x or 7.x target under any of the following conditions:
- The module supports Java EE 5.
- The module calls a 7.x runtime API.
- The module uses a 7.x product feature.

If a module supports Java 2 Platform, Enterprise Edition (J2EE) 1.4, then you can install the module on a Version 6.x or later 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, 7.x or 8.x deployment target. Modules that require 6.1.x feature pack functionality can be installed on a 7.x or 8.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, 7.x or 8.x target.

You can install an application or module developed for a Version 5.x product on any deployment 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.

Table 41. Application status. Shows whether the application is running.

Icon	Application status	Description
	Started	Application is running.
	Partial Start	Application is in the process of changing from a *Stopped* state to a *Started* 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 *Started* state to a *Stopped* 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**.

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. Version 8.x supports Java EE 6 enterprise applications and modules.

If you are deploying Java EE 6 modules, ensure that the deployment target supports Version 8.0 or later. You can deploy Java EE 6 modules only to Version 8.x and later servers. You cannot deploy Java EE 6 modules to Version 7.x or 6.x deployment targets.

The administrative console Server collection pages show the versions for 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.

Service Component Architecture (SCA) applications can contain Java EE modules as well as SCA composites. You can use the procedure described in this topic to update Java EE modules in SCA applications. However, you cannot use this procedure to update SCA composites or other SCA-specific files.

Procedure

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
   - Drag and drop to a monitored directory
   - 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. If needed, restart the application manually so the changes take effect. Start the deployed application files using
   - Administrative console
   - wsadmin startApplication
   - Java programs that use ApplicationManager or AppManagement MBeans

   When you update an application 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.

   If you update module metadata while an application is running, restarting the application might not be sufficient for the changes to take effect. For example, if you change descriptors in running Java EE 6

applications that use annotations, you must reinstall the application. If you change classes that introduce, remove, or alter class hierarchies within an application, and those changes impact annotated classes, you also must reinstall the application.

What to do next

Save the changes to your administrative configuration.

Note:

- You must use either the administrative console or wsadmin scripting to synchronization a node. Of these two options, using the administrative console is the best way to perform this operation. The Nodes panel in the administrative console includes the **Synchronize** operation.

  If you need to use wsadmin scripting to synchronize a node, use the NodeSync mbean's sync() command.

- Do not restart the node agent as part of the synchronize node process. Administration operations, such as node synchronization for application deployment, or updates that take place while the node agent is starting, that are initiated through the node agent, and affect the application servers, fail until the node agent has a chance to discover the application servers.

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 42. Ways to update application files. You can update application files using the console, wsadmin, programming, or deployment tools

Option	Method	Comments	Starting after update
Administrative console update wizard See "Updating enterprise applications with the console" on page 386. To remove a single file from a Java EE application or module, see the topic on removing enterprise files.	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**.
Monitored directory	If you installed an application or module by adding it to a monitored directory, you can update it by adding an application or module that has the same file name to the monitored directory.	Refer to the topic on installing enterprise application files by adding them to a monitored directory.	After application or module installation, the product starts it automatically.
Application properties files	Create a properties file that specifies to update application files. Then, run the wsadmin **applyConfigProperties** command in any of the following ways: • In a script • At a command prompt • By adding the properties file to a `dragDropDeployableApps/ deploymentProperties` monitored directory	Refer to the topic on installing enterprise application files by adding properties files to a monitored directory.	After application or module installation, the product starts it automatically.

Table 42. Ways to update application files (continued).   You can update application files using the console, wsadmin, programming, or deployment tools

Option	Method	Comments	Starting after update
wsadmin scripts See the topic on updating installed applications using the wsadmin scripting tool.	Use the **update** command or the **updateInteractive** command in a script or at a command prompt. For more information on the **update** and **updateInteractive** commands, see the topic on commands for the AdminApp object.	The Getting started with wsadmin scripting topic provides an overview of wsadmin.	Start the application using the **invoke** command and the startApplication attribute. For more information about the **invoke** command, see the topic on commands for the AdminControl object.
Java application programming interfaces See the topic on using administrative programs (JMX).	Update deployed applications by completing the steps in the topic on managing applications through programming.	Update an application in the following ways: • 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	• Invoke the AdminApp **startApplication** command. • Invoke the **startApplication** method on an ApplicationManager MBean using AdminControl.
Rapid deployment tools See 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.	Rapid deployment tools offer the following advantages: • 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 previous 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: 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 previous 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) archive (`.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*.

If you update module metadata while an application is running, restarting the application might not be sufficient for the changes to take effect. For example, if you change descriptors in running Java EE 6 applications that use annotations, you must reinstall the application. If you change classes that introduce, remove, or alter class hierarchies within an application, and those changes impact annotated classes, you also must reinstall the application.

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 385 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.

Procedure

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 using the console and on the Preparing for 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 or later nodes. You can deploy Java EE 6 modules to servers on Version 8.x or later nodes.

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.

   When you update a full application in the single server (base) product, after you save the changes, the old version of the application is uninstalled and the new version is installed into the configuration. The application binaries for the old version are deleted from the destination directory and the new binaries are copied to the directory.

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.

   If you update module metadata while an application is running, restarting the application might not be sufficient for the changes to take effect. For example, if you change descriptors in running Java EE 6 applications that use annotations, you must reinstall the application. If you change classes that introduce, remove, or alter class hierarchies within an application, and those changes impact annotated classes, you also must reinstall the application.

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 stand-alone WAR module, then specify the context root as well.

   > **Tip:** 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 stand-alone 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 stand-alone 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 (module URI) , 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`.

To replace a file, the relative path must match the relative path of the file to be updated in the installed application.

To add a new file to the installed application, the specified relative path must *not* match the relative path of an already existing 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. 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	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.
	For example, to delete a file named utils/config.xmi from the root of the my.ear file, include the line utils/config.xmi in the META-INF/ibm-partialapp-delete.props file.

Level of files to delete	Metadata .props file to include in compressed file
Module	Include *module_uri*/META-INF/ibm-partialapp-delete.props in the compressed file. 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. 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.

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.

Escape regular expression metacharacters in the META-INF/ibm-partialapp-delete.props file. For example, to delete inner classes for a class named Abc, use the regular expression Abc\$.* where $ is a regular expression metacharacter that is escaped with a backslash (\).

A META-INF/ibm-partialapp-delete.props file might contain the following text:

```
.*.dat
```

```
webmod.war/META-INF/ibm-partialapp-delete.props:
com/test/.*.jsp
WEB-INF/test.xmi
```

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

The following note applies to the xmi file references in this topic:

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Restriction: `Solaris` `HP-UX` The hot deployment and dynamic reloading function is not supported when the product is running on these operating systems. The Java archive (JAR) files within the associated Java Development Kit (JDK) are memory mapped. If these JAR files are updated by the hot deployment and dynamic reloading functionality when they are being used by the Java virtual machine (JVM), the files become inconsistent, which results in an application server crash. When you make changes to an application on these operating systems, do not use the hot deployment and dynamic reloading functionality. Instead, restart the application to reflect the changes.

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 385 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 384, 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.

Procedure

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.

   Important: 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 380 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

The following note applies to the xmi file references in this topic:

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Restriction: Solaris   HP-UX The hot deployment and dynamic reloading function is not supported when the product is running on these operating systems. The Java archive (JAR) files within the associated Java Development Kit (JDK) are memory mapped. If these JAR files are updated by the hot deployment and dynamic reloading functionality when they are being

used by the Java virtual machine (JVM), the files become inconsistent, which results in an application server crash. When you make changes to an application on these operating systems, do not use the hot deployment and dynamic reloading functionality. Instead, restart the application to reflect the changes.

There are several changes that you can make to deployed application files without stopping the server and starting it again.

Important: See "Ways to update enterprise application files" on page 385 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.

Table 43. Available changes to deployed application files. Available changes using hot deployment or dynamic reloading.

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 `application.xml` file for an application.	Not applicable	Yes
Change the `ibm-app-ext.xmi` file for an application.	Not applicable	Yes
Change the `ibm-app-bnd.xmi` file for an application.	Not applicable	Yes
Change a non-module Jar file contained in the EAR file.	Yes	Yes

Procedure

- 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

The following note applies to the file references with a .xmi extension in this topic:

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.

- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Restriction: Solaris HP-UX The hot deployment and dynamic reloading function is not supported when the product is running on these operating systems. The Java archive (JAR) files within the associated Java Development Kit (JDK) are memory mapped. If these JAR files are updated by the hot deployment and dynamic reloading functionality when they are being used by the Java virtual machine (JVM), the files become inconsistent, which results in an

application server crash. When you make changes to an application on these operating systems, do not use the hot deployment and dynamic reloading functionality. Instead, restart the application to reflect the changes.

There are several changes that you can make to WAR files without stopping the server and starting it again.

Important: See "Ways to update enterprise application files" on page 385 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.

Table 44. Available changes to deployed WAR files. Available changes using hot deployment or dynamic reloading.

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

Procedure

- 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.

  Note: The reloadInterval and reloadingEnabled attributes of the IBM deployment descriptor extensions, including both the WAR file extension `WEB-INF/ibm-web-ext.xmi` and the application extension `META-INF/ibm-application-ext.xmi` are deprecated.

- 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) Java archive (JAR) files on application servers without having to stop the server and start it again.

About this task

The following note applies to the file references with .xmi extensions in this topic:

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Restriction: Solaris HP-UX The hot deployment and dynamic reloading function is not supported when the product is running on these operating systems. The Java archive (JAR) files within the associated Java Development Kit (JDK) are memory mapped. If these JAR files are updated by the hot deployment and dynamic reloading functionality when they are being used by the Java virtual machine (JVM), the files become inconsistent, which results in an application server crash. When you make changes to an application on these operating systems, do not use the hot deployment and dynamic reloading functionality. Instead, restart the application to reflect the changes.

There are several changes that you can make to EJB JAR files without stopping the server and starting it again.

Important: See "Ways to update enterprise application files" on page 385 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.

Table 45. Available changes to EJB JAR files. Available changes using hot deployment or dynamic reloading.

Change	Hot deployment	Dynamic reloading
Change the `ejb-jar.xml` file of an EJB JAR file.	Not applicable	Yes
Change the `ibm-ejb-jar-ext.xmi` or `ibm-ejb-jar-bnd.xmi` file of an EJB JAR file.	Not applicable	Yes
Change the `Table.ddl` file for an EJB JAR file.	Not applicable	Not applicable
Change the `Map.mapxmi` or `Schema.dbxmi` 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

Procedure

- 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

Restriction: `Solaris` `HP-UX` The hot deployment and dynamic reloading function is not supported when the product is running on these operating systems. The Java archive (JAR) files within the associated Java Development Kit (JDK) are memory mapped. If these JAR files are updated by the hot deployment and dynamic reloading functionality when they are being used by the Java virtual machine (JVM), the files become inconsistent, which results in an application server crash. When you make changes to an application on these operating systems, do not use the hot deployment and dynamic reloading functionality. Instead, restart the application to reflect the changes.

There are several change that you can make to the HTTP plug-in configuration without stopping the server and starting it again.

Important: See "Ways to update enterprise application files" on page 385 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.

Table 46. Available changes to HTTP plug-in configuration files. Available changes using hot deployment or dynamic reloading.

Change	Hot deployment	Dynamic reloading
Change the `application.xml` file to change the context root of a web application archive (WAR file).	Yes	No
Change the `web.xml` file to add, remove, or modify a servlet mapping.	Yes	Yes
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.	Yes	Yes

Note: The following steps reference the `GenPluginCfg.bat/sh` script. You must delete the `plugin-cfg.xml` file in the *profile_root*/`config/cells` directory before you use the `GenPluginCfg.bat/sh` script. Otherwise, configuration changes do not persist to the `plugin-cfg.xml` file.

Procedure

- 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.

Resolving application configuration conflicts

In a shared environment with multiple administrative users, it is possible that different administrative users might attempt concurrent updates of the same WebSphere Application Server configuration documents. The following information should help you detect and deal with any exceptions that might occur if multiple administrative users attempt to concurrently update the same configuration documents.

Whenever you log into an administrative client using either the administrative console or wsadmin tool, a unique workspace session is created to track any configuration changes that are made. For each workspace session, a temporary workspace directory is associated with each workspace session. This directory is used to store all of the configuration files that you change during a login session. The files in this directory are initially extracted from the cell configuration repository, and your changes only exist in the workspace copies of these files until a save occurs. When a save occurs, the configuration management runtime makes sure that the configuration files you changed have not already been modified and saved by another user, and then copies the changed files from your workspace directory back to the master repository,

As long as different workspace sessions modify different configuration files, there is no save conflict. However if multiple workspace sessions modify one or more of the same configuration files, a save conflict occurs, and only the first workspace session changes are reflected in the configuration repository for the cell. When subsequent users attempt to save the changes to the same configuration files, they receive a save conflict exception

With regard to application deployment, a special provision is made for supporting the concurrent deployment of applications. Application deployment initiated by different workspace sessions can modify the same `serverindex.xml` file. In this situation, the configuration management runtime employs a merge algorithm for the `serverindex.xml` file which supports:

- Concurrent deployment of different applications to different application servers or to different application server clusters.

- Concurrent deployment of different applications to the same application server or to the same application server cluster.

Typically, concurrent or parallel application deployment scenarios are safely executed without any additional effort on the part of the system administrator. However, there are application deployment scenarios that the `serverindex.xml` merge algorithm does not handle. For example, the `serverindex.xml` merge algorithm does not handle situations where file changes are concurrently deployed to the same application, the same cluster or the same server. The merge algorithm also does not handle configuration conflicts that arise during other concurrent administration activities that involve more than application deployment.

Obtaining the requisite object references and constructing the parameter list

There are some simple measures you can take to ensure against configuration conflicts, and to resolve any conflicts when using the wsadmin tool. The wsadmin commands that are employed for configuration conflict detection and resolution rely on obtaining a reference to the ConfigService MBean, and then invoking the getConflictDocuments method provided by that MBean to determine if users have made conflicting changes to a file during their workspace session. See the Javadoc for the ConfigService MBean for more information about this Mbean.

The following code example illustrates how to obtain the requisite object references and construct the parameter list that is required to invoke the getConflictDocuments method that the ConfigService MBean provides:

```
// get ConfigService MBean reference

wsadmin>cs = AdminControl.queryNames('WebSphere:*,type=ConfigService')

// obtain ObjectName for ConfigService MBean

wsadmin>import javax.management as mgmt

wsadmin>csName=mgmt.ObjectName(cs)

// get session object for the current administrative user session

wsadmin>session=AdminConfig.getCurrentSession()

// manipulate and prepare the administrative session object and

// MBean operation arguments for use

wsadmin>from com.ibm.websphere.management import Session

wsadmin>from jarray import array

wsadmin>parms=array([session], java.lang.Object)

wsadmin>ptype=array(['com.ibm.websphere.management.Session'], java.lang.String)
```

After the variables and parameter list is initialized, the getConflictDocuments method is invoked. If there are no conflicts, the method returns the following message:

```
// invoke MBean getConflictDocuments method to obtain a list of any document conflicts

wsadmin>AdminControl.invoke_jmx(csName,'getConflictDocuments', parms, ptype)
{}
wsadmin>
```

If configuration conflicts exist because of changes another user made during a work session, the method returns a message, similar to the following message, that lists the XML files that have changed:

```
Listing 3
wsadmin>AdminControl.invoke_jmx(csName,' getConflictDocuments', parms, ptype)
{['cells/cell_name/nodes/node_name/serverindex.xml',cells/cell_name/applications/
DefaultApplication.ear.ear/deltas/DefaultApplication.ear/delta-1278791909117',
... <list abbreviated> ...}

    wsadmin>
```

In this situation, you can issue the AdminConfig.reset() command to discard the changes that you made since the last AdminConfig.save() command was issued:

```
wsadmin>AdminConfig.reset()
```

Even if you call the getConflictDocuments method prior to saving your changes, and see that there are no conflict documents, there is no guarantee that a save will succeed, even if you immediately issue the AdminConfig.save() command because some other session might have modified the same configuration files between when you call the getConflictDocuments method, and when you issue the AdminConfig.save() command.

When a save to the master repository is unsuccessful, you get a ConfigServiceException exception that is similar to the following exception:

```
WASX7015E: Exception running command: "AdminConfig.save()"; exception information:
com.ibm.websphere.management.exception.ConfigServiceException
java.security.PrivilegedActionException:
java.security.PrivilegedActionException:
com.ibm.ws.sm.workspace.WorkSpaceException: RepositoryException
```

If you receive a save conflict exception:

1. Use the getConflictDocuments method to determine which configuration files were already saved by another user.
2. Issue the AdminConfig.reset() command to discard your changes.
3. After discarding these changes, you can reapply your changes to the appropriate configuration files, and issue the AdminConfig.save() command to save these changes.

The subsequent attempt to save your changes is likely to succeed because it is not typical to experience multiple save conflicts during the same session. However, if this subsequent save attempt is not successful, repeat the preceding actions, and save your changes again.

Exporting enterprise applications

You can export an enterprise application to a location of your choice.

Before you begin

This topic assumes that you have installed an enterprise application on a server and that you want to export the application.

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.

Procedure

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.
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 405. The exported enterprise archive (EAR) file contains application configuration data as well as the application.

Procedure

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.

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.

Procedure

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.

   Tip: 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.

Uninstalling enterprise applications using the console

After an application no longer is needed, you can uninstall it.

Before you begin

This topic assumes that you have installed an enterprise application on a server and that you want to delete the application from the server.

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.

Procedure

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 page, 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.

Before you begin

This topic assumes that you have installed an application or module file on a server and that you want to delete the file.

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.

Procedure

- 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 stand-alone 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.

Uninstalling enterprise application files by dragging them from a monitored directory

After an application file no longer is needed, you can uninstall it by removing its enterprise archive (EAR), web application archive (WAR), Java archive (JAR), or Session Initiation Protocol (SIP) archive (SAR) from a monitored directory. An enterprise application file must conform to the Java Platform, Enterprise Edition (Java EE) specification.

Before you begin

This topic assumes that you previously installed an enterprise application or module on a server by dragging or copying it to a monitored directory and that you want to delete it from the server. The EAR, JAR, WAR, or SAR file exists in a monitored subdirectory:

- *app_server_root*/profiles/*profile_name*/monitoredDeployableApps

For base (stand-alone) application servers on distributed operating systems, the monitored directory is under the application server profile. The directory path is *app_server_root*/profiles/ *application_server_profile_name*/monitoredDeployableApps/servers/*server_name*.

Restriction: Installing an EAR, JAR, WAR, or SAR file by adding it to a monitored directory is available only on distributed and z/OS operating systems. It is not supported on IBM i operating systems.

About this task

For applications previously installed by adding an EAR, JAR, WAR, or SAR file to a monitored monitoredDeployableApps subdirectory, you can uninstall the application or module from the application server by dragging or deleting the EAR, JAR, WAR, or SAR file from the monitoredDeployableApps subdirectory.

The product scans a monitored directory for new applications no more frequently than every five seconds, by default. The server must be running for the product to notice changes to files in its monitored directory.

After not finding the EAR, JAR, WAR, or SAR file in the monitored directory, the product stops the application or module and uninstalls the file from the application server. Also, the product removes application files from the directory for installed applications:

- *app_server_root*/profiles/*profile_name*/installedApps/*cell_name*

Uninstalling an application file deletes it from the product configuration repository and deletes the application module binaries from the file system of the node where the application modules are installed.

This topic assumes that you use a graphical file browser to drag or delete the EAR, JAR, WAR, or SAR file from the directory. Alternatively, you can use operating system commands to delete a file from a monitored directory.

Procedure

1. Ensure that the application server from which you want to remove the enterprise application file is running.

   To see which servers are running, you can use the serverStatus -all command. To start a server, you can use the startServer *server_name* command.

   For example, for a stand-alone application server node with a profile name of AppSrv02, run the serverStatus command from a command prompt at the *app_server_root*/profiles/AppSrv02/bin directory:

   serverStatus -all

   If the server is not running, start the server. For example, to start an application server named server1 on AppSrv02, run the startServer command from a command prompt at the *app_server_root*/ profiles/AppSrv02/bin directory:

   startServer server1

   The Server server1 open for e-business message indicates that the server is running.

2. Open a file browser and locate the monitored directory.

   For example, for server1 in the AppSrv02 profile, locate the *app_server_root*/profiles/AppSrv02/ monitoredDeployableApps/servers/server1 directory.

3. Remove the EAR, JAR, WAR, or SAR file that you want to uninstall.

   For example, drag or delete the DynaCacheEsi.ear file from the *app_server_root*/profiles/AppSrv02/ monitoredDeployableApps/servers/server1 directory.

Results

The product stops the application or module and uninstalls it from the application server.

For the AppSrv02 example, the product writes messages about the uninstallation to the `SystemOut.log` file in the `app_server_root`/logs/`server_name` directory. The messages start with the `CWLDD` message key.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

What to do next

Examine the server to verify that the application or module is no longer installed.

If the uninstallation is not successful, read messages in the `SystemOut.log` and fix the error condition. Add the EAR, JAR, WAR, or SAR file back into the monitored `monitoredDeployableApps` subdirectory, wait for the product to detect the file, and then remove it from the directory again. If the uninstallation still is not successful, use the administrative console or a wsadmin script to uninstall the application file.

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 412
- "Administration" on page 412

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://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html`
- Recommended reading list: Java EE 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 V8 Programming Guide, SG24-7835-00, `http://www.redbooks.ibm.com/abstracts/sg247835.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 V7 Administration and Configuration Guide, SG24-7615-01, `http://www.redbooks.ibm.com/abstracts/sg247615.html`

Chapter 10. 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.

Procedure

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 Chapter 11, "Extending application management operations through programming," on page 463. 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.

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.

Procedure

- 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.

Preparing an application for installation using programming

The product application management architecture provides a set of classes that allows application developers to collect WebSphere Application Server-specific deployment information, hereafter called binding information, and store it in the application EAR file. Such an EAR file can then be installed into a WebSphere Application Server configuration, using application management interfaces that are described in the topic Installing an application through programming. This topic uses a programming example to explain how an EAR file can be populated with binding information.

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

Complete the following tasks to prepare an application for installation through programming.

Procedure

1. Create an AppDeploymentController.

   The AppDeploymentController takes an EAR file as an argument and creates a sequence of steps, represented by AppDeploymentTask objects (hereafter called tasks). The tasks are typically presented to the deployer to collect binding information, or are manipulated programmatically. The following code excerpt shows how to create an AppDeploymentController instance:

```
// create preferences to pass in the locale information
Hashtable prefs = new Hashtable();
prefs.put(AppConstants.APPDEPL_LOCALE, Locale.getDefault());
```

```
// create a controller instance for EAR file
// specified by a fully-qualified path in earName
AppDeploymentController flowController =
AppManagementFactory.readArchive (earName, prefs);
```

2. Obtain AppDeploymentTask instances. After creating AppDeploymentController, you can iterate over
   the task objects that it creates. Each task collects a specific kind of binding information for various
   modules in the application or for the application itself. A task can have the following attributes, and
   corresponding getters/setters, as applicable. See the javadoc for the AppDeploymentTask API for more
   information about these attributes.

*Table 47. Task attributes. The following table describes the attributes that an AppDeploymentController task might
have.*

Name	Description	Value
appController	A controller instance that manages tasks.	AppDeploymentController
colNames	The task column names	java.lang.String[]
hasHiddenColumns	Specifies whether the task has any hidden columns.	boolean
HiddenColumns	An array of boolean that indicates that these columns should not be shown by the tool that displays the tasks to the end user.	boolean[]
HIGHEST_VERSION	Public static final java.lang.String HIGHEST_VERSION	static java.lang.String
isSufficientlyDone	Specifies whether the task has any required data that is not specified.	boolean
isTaskDisabled	Specifies whether the task should be shown or changed.	boolean
isTaskEmpty	Specifies whether the task has any data in it.	boolean
isValidationEnabled	Specifies whether the task should be validated	boolean
mutables	An array of boolean indicating if the task data in a specific column index can be changed by the person who is deploying the application.	boolean[]
name	A unique task name	java.lang.String
requiredColumns	An array of boolean indicating if a task column must have a non-null value.	boolean[]
taskData	A 2-dimensional array (table) of strings. The first row of the table contains the column headings for the task (e.g. name of the module, module URI, JNDI name etc). The rest of the rows represent application-specific information.	java.lang.String[]

Table 47. Task attributes (continued). The following table describes the attributes that an AppDeploymentController task might have.

Name	Description	Value
taskValidateErrorMessages	The error messages that are generated when a task is validated	java.lang.String[]
VERSION_HIGHEST	public static final int VERSION_HIGHEST	static int

The following table lists various task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. You might notice more tasks if the application contains WebSphere Application Server enterprise extensions.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names
MapRolesToUsers	Allows you to specify users or groups for a security role. You must map each role that is defined in the application or module to a user or group from the domain user registry. Each row of task data represents a single security role. You can specify multiple users or groups for a single role by separating them with a quotation mark (").	• role- lists the specific capabilities that are given to a user. Role privileges give users and groups permission to run as specified. For example, you might map the user Joe to the administrator role, which enables user Joe to perform all of the tasks associated with the administrator role. The authorization policy is only enforced when global security is enabled. • everyone - specifies whether to map everyone to a specified role. • allAuthenticatedUsers - specifies whether to map all authenticated users regardless of the realm to a specified role • mappedUsers - lists the users that are mapped to the specified role within this application. • • mappedGroups - lists the groups that are mapped to this specified role within this application. • allAuthenticatedUsersInRealms - specifies whether to map all users in the trusted realms to a specified role. • userAccessIds - specifies the user information that is used for Java Platform, Enterprise Edition authorization when using the WebSphere Application Server default authorization engine. The format for the user accessIds is user:realm/uniqueUserID. • groupAccessIds - specifies the group information. The format for the group accessIds is group:realm/uniqueGroupID.
MapRunAsRolesToUsers	Allows you to specify credentials for a run-as role. The enterprise beans that you install contain predefined RunAs roles. Enterprise beans that need to run as a particular role for recognition while interacting with another enterprise bean use RunAs roles. Each row of task data represents a single run-as role.	• role - maps specific capabilities to a user. The authorization policy is only enforced when global security is enabled. • userName - specifies a user name for the RunAs role user. This user already maps to a role specified for the MapRolesToUsers task. You can map the user to its appropriate role by either mapping the user to that role directly or mapping a group that contains the user to that role. • password - specifies the password for the RunAs user.
BindJndiForEJBNonMessageBinding	Allows you to specify JNDI names for enterprise java beans (EJBs). This JNDI name is used to look up EJB Homes from client programs. A row of task data specifies a single EJB for which a JNDI name can be supplied.	• EJBModule - specifies the EJB module that contains the enterprise beans that bind to the JNDI name. • • EJB - specifies the name of an enterprise bean that is contained by the module. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • JNDI - specifies the JNDI name associated with the enterprise bean in an EJB module.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names
BindJndiForEJBMessageBinding	Allows you to specify a listener port name or JNDI of an activation specification for message-driven beans (MDBs). Ensure each MDB in your application or module is bound to a listener port name or JNDI of an activation specification. Each row of task data represents a single message-driven bean.	• EJBModule - specifies the Enterprise JavaBeans module that contains the enterprise bean. • • EJB - specifies the name of an MDB in the application. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • listenerPort - specifies a listener port namefor the MDB. • actSpecJNDI - specifies an activation specification JNDI name for the MDB. When a MDB is bound to an activation specification JNDI name you can also specify the destination JNDI name and the authentication alias. • actSpecDestinationJNDI - specifies a destination JNDI name for the activation specification. • actSpecAuth - specifies a authentication alias that is used to access the user name and password that are set on the configured J2C activation specification.
BindJndiForEJBBusiness	Allows you to specify JNDI name bindings for each enterprise bean with a business interface in an EJB module. Each enterprise bean with a business interface in an EJB module must be bound to a JNDI name. A row of task data specifies a single enterprise bean for which a JNDI name can be supplied.	• EJBModule - specifies the EJB module that contains the enterprise beans that bind to the JNDI name. • • EJB - specifies the name of an enterprise bean that binds to the JNDI name.. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • businessInterface - specifies the enterprise bean business interface in an EJB module. • JNDI - specifies the JNDI name associated with the enterprise bean business interface in an EJB module.
MapEJBRefToEJB	Allows you to specify JNDI names of EJBs that are bound to ejb-refs. Each row of task data represents a single ejb-ref defined in a module, An EJB 3.0 module cannot contain container-managed or bean-managed persistence entity beans. Installation fails when a container-managed or bean-managed persistence entity bean is packaged in a EJB 3.0 module of a Java EE application. You can only package container-managed or bean-managed persistence in an EJB 2.1 or earlier module. If the EJB reference is from an EJB 3.0, Web 2.4, or Web 2.5 module, the JNDI name is optional. If the Allow EJB reference targets to resolved automatically option is enabled, the JNDI name is optional for all modules. The runtime provides a container default or automatically resolves the EJB reference if a binding is not provided.	• EJB - specifies the name of an enterprise bean that is contained by the module. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • referenceBinding - specifies the name of the EJB reference that is used in the enterprise bean. • class - Specifies the name of a Java class associated with this enterprise bean. • JNDI - specifies the JNDI name associated with the enterprise bean in an EJB module

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names

MapResRefToEJB	Allows you to specify JNDI names of resources defined in WebSphere Application Server configuration that are bound to resource-refs. Each row of task data represents a single resource-ref defined in a module.	• AppVersion - specifies the version of the application.
		
		• ModuleVersion - specifies the version of the module.
		
		• Module - specifies the name of a module in the application.
		
		• EJB - specifies the name of an enterprise bean that contains the resource reference. (Only applies for an EJB module)
		
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		
		• referenceBinding - specifies the name of a resource reference that is contained by the module.
		
		• targetResourceJNDI - specifies the JNDI name of the resource that is the mapping target of the resource reference.
		
		• Login configuration - this column applies to data sources and connection factories only, and refers to the authentication method for securing the resource. Java 2 Connectors (J2C) use the DefaultPrincipalMapping login configuration to map users to principals that are defined in the J2C authentication data entries. If the login.config.name is set to DefaultPrincipalMapping, a property is created with the name com.ibm.mapping.authDataAlias . The value of the property is set by the authentication props. If the login.config name is not set to DefaultPrincipalMapping, the authentication props can specify multiple properties.
		
		• authentication properties - specifies properties used by login configuration. The string format is:
		
		  websphere:name= \<name1>,value= \<value1>,description=\<desc1>
		
		  You can use the plus sign (+) to specify multiple properties.
		
		• Resource authorization - specifies the authorization type for securing the resource.
		
		• Extended data source properties - specifies the extended data source properties for a DB2 database. You can use these properties to allow an application to extend the custom properties for a data source, or override any non-core properties that already exist for that data source.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names
MapResEnvRefToRes	Allows you to specify the JNDI names of resources defined in the WebSphere Application Server configuration that are bound to resource-env-refs. Each row of task data represents a single resource-env-ref defined in a module.	• Module - specifies the name of a module in the application. • EJB - specifies the name of an enterprise bean that contains the resource environment reference. (Only applies for an EJB module). • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • referenceBinding - specifies the name of a resource environment reference. The reference corresponds to a resource that is bound as a resource environment entry into the JNDI name space of the application server. • resEnvRef.type - specifies the type associated with the resource environment reference. • JNDI- specifies the JNDI name of the resource environment entry that is the mapping target of the resource environment reference.
MapSharedLibForMod	Allows you to associate defined shared libraries with an application or Web module. A shared library is an external Java archive (JAR) file that is used by one or more applications. Using shared libraries enables multiple applications deployed on a server to use a single library, rather than use multiple copies of the same library. After you associate shared libraries with an application or module, the application or module class loader loads classes represented by the shared libraries and makes those classes available to the application or module. Each row of task data represents a single application or Web module.	• application - specifies the name of the application that you are installing. • module - specifies the name of the module associated with the shared libraries. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • sharedLibraries - Specifies the name of the shared library files associated with the application or module

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names

SharedLibRelationship	Allows you 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. Each row of task data represents a single application or Web module.	• module - specifies the name of the module associated with the shared libraries.
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		• relationshipIdentifiers - specifies a shared library asset or composition unit identifier for the module shared library relationship. The string format is:
		  assetname=<assetName> or cuname=<cuName>
		  You can use the plus sign (+) to specify multiple relationships.
		• compositionUnit - specifies the 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. Composition unit name is positionally matched with the name of the asset or composition unit identifier specified in the relationship field.
		• matchTarget - specifies whether the product maps the composition unit for the shared library relationship to the same deployment target as the business-level application. The following values are valid for this column:
		  – AppConstants.YES_KEY, which indicates that you want the product to maps the composition unit for the shared library relationship to the same deployment target as the business-level application.
		  – AppConstants.NO_KEY, which indicates that you do not want the product to maps the composition unit for the shared library relationship to the same deployment target as the business-level application.

CtxRootForWebMod	Allows you to specify the context root for Web modules during or after installation of an application onto a WebSphere Application Server deployment target. Each row of task data represents a single Web module.	• webModule - specifies the name of a Web module in the application that you are installing or that you are viewing after installation. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • contextRoot - specifies the context root of the Web application (WAR). A context root for each Web module is defined in the application deployment descriptor during application assembly. 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.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names

MapInitParamForServlet	Allows you to specify initial parameter values for servlets in Web modules during or after installation of an application onto a WebSphere Application Server deployment target. The `<param-value>` values specified in `<init-param>` statements in the web.xml file of Web modules are used by default. Each row of task data represents a single servlet.	• module - specifies the name of a module in the application that you are installing or that you are viewing after installation.
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		• servlet - specifies a unique name for the servlet within the application. You must package servlets in a Web archive (WAR) file or Web module for deployment to an application server. Servlets run on a Java-enabled Web server and extend the capabilities of a Web server, similar to the way applets run on a browser and extend the capabilities of a browser.
		• name - specifies the name of the initial parameter passed to the init method of the Web module servlet filter. The following example servlet filter statement in a web.xml file specifies an initial parameter name of attribute:
		
		`<init-param>`
		`    <param-name>attribute</param-name>`
		`    <param-value>tests.Filter.`
		`DoFilter_Filter.SERVLET_MAPPED`
		`    </param-value>`
		`</init-param>`
		
		• value - specifies the value assigned to an initial parameter passed to the init method of the Web module servlet filter. The following example servlet filter statement in a web.xml file specifies an initial parameter value of tests.Filter.DoFilter_Filter.SERVLET_MAPPED for the init parameter attribute:
		
		`<init-param>`
		`    <param-name>attribute</param-name>`
		`    <param-value>tests.Filter.`
		`DoFilter_Filter.SERVLET_MAPPED`
		`    </param-value>`
		`</init-param>`
		
		• description - specifies information about the initial parameter.

MapEnvEntryForEJBMod	Allows you to configure the environment entries of Enterprise JavaBeans™ (EJB) modules such as entity, session, or message driven beans. Each row of task data represents a single environment entry in an EJB module.	• module - specifies the name of an EJB module. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • EJB - specifies the name of an enterprise bean that is contained by the module. • name - specifies the name of the environment entry that you are editing or viewing. The environment entry is the env-entry property in the EJB module. • type - specifies a data type for the environment entry defined by the env-entry property in the EJB module. • description - specifies information on the environment entry. • value - specifies an editable value for the environment entry defined by the env-entry property in the EJB module.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names
MapEnvEntryForWebMod	Allows you to configure the environment entries of Web modules such as servlets and JavaServer Pages (JSP) files. Each row of task data represents a single environment entry in a Web module.	• module - specifies the name of an Web module. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • name - specifies the name of the environment entry that you are editing or viewing. The environment entry is the env-entry property in the Web module. • type - specifies a data type for the environment entry defined by the env-entry property in the Web module. • description - specifies information on the environment entry. • value - specifies an editable value for the environment entry defined by the env-entry property in the Web module.
MapMessageDestinationRefToEJB	Allows you to specify the JNDI name of the J2C administered object to bind the message destination reference to the message driven beans. Each row of task data represents a single message destination reference in a module.	• module - specifies the name of an EJB module. • EJB - specifies the name of message-driven beans (MDBs) that is contained by the module. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • messageDestinationObject - specifies the name of the message destination reference or message destination link if a destination link is provided • JNDI - specifies the target JNDI of the referenced message destination bean. If a message destination link is provided, this field defaults to the destination JNDI of the message driven bean that the message destination link references.
DataSourceFor10EJBModules	Allows you to specify JNDI name of the default data source used for an EJB module that contains EJB1.x beans. Each row of task data represents a single EJB module.	• EJBModule - specifies the name of the module that contains the 1.x enterprise beans. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • JNDI - specifies the JNDI name default data source for the EJB module. • userName - specifies the user name that, along with the password, comprises the authentication alias for securing the data source. • password- specifies the password that, along with the user name, comprises the authentication alias for securing the data source. • Login configuration - specifies to the authentication method for securing the data source. • authentication properties - specifies properties used by login configuration. The string format is: `websphere:name= <name1>,value=<value1>, description=<desc1>` You can use the plus sign (+) to specify multiple properties.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names

DataSourceFor20EJBModules	Allows you to specify JNDI name of the default connection factory used for an EJB module that contains EJB2.x beans. Each row of task data represents a single EJB module.	• EJBModule - specifies the name of the module that contains the 1.x enterprise beans.
		
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		
		• JNDI - specifies the JNDI name default data source for the EJB module.
		
		• Resource authorization - specifies the authorization type for securing the resource. The valid values are
		
		`AppConstants.APPDEPL_CMPBINDING_ RESAUTHTYPE_CONTAINER`
		
		and
		
		`AppConstants.APPDEPL_CMPBINDING RESAUTHTYPE_PER_CONNECTION_FACTORY`
		
		.
		
		• Login configuration - specifies to the authentication method for securing the data source.
		
		• authentication properties - specifies properties used by login configuration. The string format is:
		
		`websphere:name= <name1>,value=<value1>, description=<desc1>`
		
		You can use the plus sign (+) to specify multiple properties.
		
		• Extended data source properties - specifies the extended data source properties for a DB2 database.
		
		You can use these properties to allow an application to extend the custom properties for a data source, or override any non-core properties that already exist for that data source.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names

DataSourceFor10CMPBeans	Allows you to specify JNDI name of the data source to be used for an EJB1.x bean with container managed persistence. A row of task data represents a single EJB.	• EJBModule - specifies the name of the module that contains the 1.x enterprise beans.
		• EJB - specifies the name of an enterprise bean that is contained by the module.
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		• JNDI - specifies the JNDI name default data source for the EJB module.
		• userName - specifies the user name that, along with the password, comprises the authentication alias for securing the data source.
		• password- specifies the password that, along with the user name, comprises the authentication alias for securing the data source.
		• Login configuration - specifies to the authentication method for securing the data source.
		• authentication properties - specifies properties used by login configuration. The string format is:
		`websphere:name=<name1>,value=<value1>,`
		`description=<desc1>`
		You can use the plus sign (+) to specify multiple properties.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names

DataSourceFor20CMPBeans	Allows you to specify JNDI name of the connection factory to be used for an EJB2.x bean with container managed persistence. A row of task data represents a single EJB.	• EJBModule - specifies the name of the module that contains the 1.x enterprise beans.
		
		• EJB - specifies the name of an enterprise bean that is contained by the module.
		
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		
		• JNDI - specifies the JNDI name default data source for the EJB module.
		
		• Resource authorization - specifies the authorization type for securing the resource. The valid values are
		
		`AppConstants.APPDEPL_CMPBINDING_`
		`RESAUTHTYPE_CONTAINER`
		
		and
		
		`AppConstants.APPDEPL_CMPBINDING`
		`RESAUTHTYPE_PER_CONNECTION_FACTORY`
		`.`
		
		• Login configuration - specifies to the authentication method for securing the data source.
		
		• authentication properties - specifies properties used by login configuration. The string format is:
		
		`websphere:name= <name1>,value=<value1>,`
		`description=<desc1>`
		
		You can use the plus sign (+) to specify multiple properties.
		
		• Extended data source properties - specifies the extended data source properties for a DB2 database.
		
		You can use these properties to allow an application to extend the custom properties for a data source, or override any non-core properties that already exist for that data source.

MapWebModToVH	Allows you to specify virtual hosts for Web modules. Each row of task data represents a Web module in the application for which virtual host information is to be collected. Web modules can be installed on the same virtual host or dispersed among several virtual hosts. Each row of task data represents a single Web module.	• webModule - specifies the name of a Web module in the application that you are installing or that you are viewing after installation. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • virtualHost - specifies the name of the virtual host to which the Web module is currently mapped. 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.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names

EnsureMethodProtectionForI0EJB	Allows you to specify if all unprotected methods of an EJB1.x module should be made inaccessible. Each task row represents a single EJB1.x module.	• EJBModule - specifies the name of the module that contains the EJB2.x beans.
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		• method.denyAllAccessPermission - specifies one of the following access permissions:
		  – AAppConstants.YES_KEY, which indicates that you want to protect this EJB module by making it inaccessible to users regardless of their access permissions.
		  – AppConstants.NO_KEY, which indicates that you want to make this EJB module accessible to users that have the appropriate access permissions.

EnsureMethodProtectionFor20EJB	Allows you to specify protection level for unprotected methods of EJB2.x beans in EJB modules. Each row of task data specifies method protection per EJB module.	• EJBModule - specifies the name of the module that contains the EJB2.x beans.
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		• method.protectionType - specifies one of the following protection methods:
		  – AppConstants.APPDEPL_METHOD_PROTECTION_UNCHECK, which indicates that you do not want the application server to verify the access permissions for the EJB module. Everyone can access the EJB module.
		  – AppConstants.APPDEPL_METHOD_PROTECTION_EXCLUDE, which indicates that you want to protect this EJB module by making it inaccessible to users regardless of their access permissions.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names
AppDeploymentOptions	Allows you to specify various deployment-specific options. This task has only 2 rows. The first row specifies the option name and the second row has the corresponding option value. The following values are the valid values for these options unless otherwise indicated in the option description: • AppConstants.YES_KEY, which indicates you want to use this deployment option. • AppConstants.NO_KEY, which indicates that you do not want to use this deployment option.	The options names that can be specified are: • preCompileJSPs - indicates whether the JavaServer Pages files are recompiled. • installed.ear.destination - specifies the directory to which you want the enterprise archive (EAR) file installed. The value for this option is a fully-qualified directory path such as *profile_root*/installedApps/mycell/myapp.ear • distributeApp - indicates whether the application management component distributes application binaries. • useMetaDataFromBinary - indicates whether the metadata that is used at run time, such as deployment descriptors, bindings, and extensions, come from the EAR file. • deployejb - specifies whether to run the EJBDeploy tool during installation. • appname - specifies the name of the application you are deploying with these options. • createMBeansForResources - indicates whether MBeans are created for all resources, such as servlets, JavaServer Pages (JSP) files, and enterprise beans, that are defined in an application when the application starts on a deployment target. • reloadEnabled - indicates whether the file system of the application is scanned for updated files so that changes reload dynamically. • reloadInterval - specifies, in seconds, the length of time that the file system of the application will be scanned for updated files. The value specified for this option is an integer greater than zero. • eployws - specifies whether to run the wsdeploy tool during deployment. • validateinput - 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. Valid values are AppConstants.APPDEPL_VALIDATE_INSTALL_OFF, AppConstants.APPDEPL_VALIDATE_INSTALL_WARM, and AppConstants.APPDEPL_VALIDATE_INSTALL_FAIL. • processEmbeddedConfig - specifies whether the embedded configuration should be processed. An embedded configuration consists of files such as resource.xml and variables.xml. • filepermission - specifies access permissions for application binaries for installed applications that are expanded to the specified directory. • blaname - 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 • autolink - specifies whether the product assigns default JNDI values for any incomplete EJB reference targets, or automatically resolves any incomplete EJB reference targets.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names
JSPReloadForWebMod	Allows you to configure the class reloading of Web modules such as JavaServer Pages (JSP) files and to select a JSF implementation to use with this application. Each row of task data represents a single Web module.	• webModule - specifies the name of a Web module in the application that you are installing or that you are viewing after installation. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • enableJSPClassReloading - specifies whether to enable class reloading when JSP files are updated. • JSPReloadInterval - specifies, in seconds, how frequently the product scans the file system for the application to check for updated JSP files.

EJBDeployOptions	Allows you 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.	• deployejb.classpath - specifies the class path of one or more zipped or Java archive (JAR) files on which the JAR or EAR file being installed depends.
		• deployejb.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. The following values are valid for this column:
		  – AAppConstants.YES_KEY, which enables the EJB deployment tool to pass RMIC options to the Remote Method Invocation compiler.
		  – AppConstants.NO_KEY, which prevents the EJB deployment tool from passing RMIC options to the Remote Method Invocation compiler.
		• deployejb.dbtype - specifies the name of the database vendor, which is used to determine database column types, mapping information, Table.sql, and other information. 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, do no specify a value for this column.
		• deployejb.dbschema - specifies the name of the schema that you want to create. he 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"
		
		• deployejb.dbaccesstype - 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.
		• deployejb.SQLJclasspath - specifies the classpath 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).
		• deployejb.JDKcompliancelevel - specifies the Java developer kit compiler compliance level when you include application source files for compilation.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names

JSPCompileOptions	Allows you to specify various options to be used by the JavaServer Pages (JSP) compiler.	• webModule - specifies the name of a Web module in the application.
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		• JSPclasspath - specifies a temporary classpath for the JSP compiler to use when compiling JSP files during application installation. This classpath is not saved when the application installation is complete and is not used when the application is running. This classpath is used only to identify resources outside of the application which are necessary for JSP compilation and which will be identified by other means (such as shared libraries) after the application is installed.
		
		In network deployment configurations, this class path is specific to the deployment manager machine.
		• useFullPackageNames - specifies whether the JSP engine generates and loads JSP classes using full package names. When full package names are used, precompiled JSP class files can be configured as servlets in the web.xml file, without having to use the jsp-file attribute. When full package names are not used, all JSP classes are generated in the same package, which has the benefit of smaller file-system paths.
		• JDKSourceLevel - specifies the source level at which the Java compiler compiles JSP Java sources. When specifying the value for this column, omit the decimal point in the level number. For example, specify 15 for JDK level 1.5.
		• disableJSPRuntimeCompilation - specifies whether a JSP file should never be translated or compiled at run time, even when a .class file does not exist. The following values are valid for this column:
		  – AAppConstants.YES_KEY, which indicates that you want to disable JSP runtime compilation.
		  – AppConstants.NO_KEY, which indicates that you do not want to disable JSP runtime compilation.

MapModulesToServers	Allows you to specify the target servers or clusters for modules in an application. Each row of task data represents one module in the application. The server target is specified as WebSphere:cell=cellName,node=nodeName,server=serveName, and the cluster target is specified as WebSphere:cell=cellName,cluster=clusterName. Multiple targets can be specified for a given module by delimiting them with "+". Each row of task data represents a single module.	• Module - specifies the name of a module in the application. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • server- specifies the target server or cluster for this module.
CorrectUseSystemIdentity	Allows you to manage the system identity properties for the Enterprise JavaBeans™ (EJB) method in your application.	• role - specifies the RunAs role that is used for this EJB method. • userName - specifies a user name for the RunAs role user. The user name is used in conjunction with the RunAs role that you select for the role. • password - specifies the password that is associated with the user name in the user registry.

Table 48. Task names that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application (continued). The following table provides the names of the tasks that are typically created by the AppDeploymentController for a J2EE 1.2, J2EE 1.3, or a Java EE 5 or later application. It also provides a description of each tasks and the task column names.

AppDeploymentTask name	Description	Task column names

CorrectOracleIsolationLevel	Allows you to specify the isolation level for the Oracle type provider for resource references that map to resources that are using an Oracle database that is doing backend processing. Each row represents one resource reference that maps to an Oracle database resource.	• Module - specifies the name of a module in the application.
		• resourceRef - specifies the name of a resource reference
		• JNDI - the JNDI name of the resource that is the mapping target of the resource reference.
		• isolationLevel - specifies the isolation level you want to use for the Oracle type provider. The two values that can be specified for this column are:
		  – rc, which indicates a JDBC Read Committed isolation level.
		  – s, which indicates a JDBC Serializable isolation level.

ActSpecJNDI	Allow you to provide JNDI names for JCA objects of embedded resource adapter modules.	• Module - specifies the name of a module in the application. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • JCA object - specifies the name of a JCA object for J2C connection factory, administered object, or activation specification. • JNDI- specifies the JNDI name for the JCA object.
BackendIdSelection	Allows you to change the backend ID for the enterprise bean Java archive (JAR) modules that have container-managed persistence (CMP) beans. An enterprise bean JAR module can support multiple backend configurations as specified using an application assembly tool. This task is useful if your application has EJB modules for which deployment code has been generated for multiple backend databases using an assembly tool.	• Module - specifies the name of an EJB module in the application. • uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR. • ID - the existing backend ID that represents a backend database. • currentID - specifies the new backend ID representing the backend database to be used

MetadataCompleteForModules	Allows each EJB 3.0 module or Web 2.5 module to write out the complete deployment descriptor including deployment information from annotations. Then the system marks the deployment descriptor for the module as complete. Each row of task data represents a single module.	• module - specifies the name of an EJB 3.0 or Web 2.5 module.
		• uri - the Uniform Resource Identifier (URI) specifies the location of the module archive relative to the root of the application EAR.
		• metadataComplete- specifies whether to write the complete module deployment descriptor including deployment information from annotation to XML format. When the metadata complete of a module is set to true, WebSphere writes out the complete deployment descriptor including deployment information from annotations. The following values are valid for this column:
		  – AppConstants.YES_KEY, which indicates that EJB 3.0 modules or Web 2.5 modules can write out the complete deployment descriptor including deployment information from annotations.
		  – AppConstants.NO_KEY, which indicates that EJB 3.0 modules or Web 2.5 modules cannot write out the complete deployment descriptor including deployment information from annotations.

The URI column in various tasks uniquely identifies a module in the application using the format as its value, where moduleURI is the module file name defined in the application's deployment descriptor, and ddURI is the URI of the deployment descriptor within the module (for standard deployment descriptor) or the URI of the alternate deployment descriptor for the module, as defined in the application's deployment descriptor. For example, if an application has a Web module in MyWeb.war and the module uses a standard deployment descriptor, then the value of the URI column in various tasks for this module is MyWeb.war,WEB-INF/web.xml. Thus the URI column value always guarantees a unique identification of a module.

The following code excerpt shows how to obtain tasks from AppDeploymentController:

```
AppDeploymentTask task =flowController.getFirstTask();
while (task != null)
{
// manipulate task data as necessary
task = flowController.getNextTask();
}
```

3. Manipulate task data

Using task name, task column names, and the J2EE artifact for which binding information is to be supplied, the task data can be modified if the corresponding column is marked as mutable. The following sample code shows how binding information can be supplied for a specific task. In this example, we are specifying users for a security role: consider that the application has a security role named Role1, and that you need to assign users User1 and User2 to that role:

```
if (task.getName().equals ("MapRolesToUsers") && !task. isTaskDisabled())
{
// find out column index for role and user column
// refer to the table above 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);
}
```

Similar logic can be used to specify all other types of binding information, such as JNDI names for EJBs, virtual host names for Web modules, etc., in various tasks. The task information and the sample code above do not allow you to collect binding information for application client modules. You need to use the Client Configuration tool shipped with WebSphere Application Server to configure application clients.

4. Save the EAR file and obtain install options.

After all the necessary binding information has been supplied in various tasks, the task data should be saved back into the EAR file so that the populated EAR file can be installed into the WebSphere

Application Server configuration. In addition to the binding information that is stored in the EAR file, there are several other deployment options that are collected by tasks, such as AppDeploymentOptions, EJBDeployOptions, and MapModulesToServers. These options are not saved in the EAR file but should be passed when the application is installed into the WebSphere Application Server configuration. After the task data is manipulated as necessary and the EAR file is saved, these options can be obtained from AppDeploymentController as a hash table. Alternately, these options can be passed directly during application installation, as explained in the Installing applications section below, and in the javadoc for installApplication API of com.ibm.websphere.management.application.AppManagement interface.

The following code shows how to save the task data into the EAR file and get generated installation options:

```
// the following line of code saves the task data
// in the EAR file specified as earName in step 1
flowController.saveAndClose();


// get the installation options
Hashtable options = flowController.getAppDeploymentSavedResults();
```

What to do next

Use programming to install the application.

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. For information on MBean programming, see MBean Java application programming interface (API) documentation.

Before you can install an application on WebSphere Application Server, you must install the application.

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.

Procedure

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_DFLTBNDG, defaultBnd);
  defaultBnd.put (AppConstants.APPDEPL_DFLTBNDG_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 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);
        }
    }
}
```

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.

Application management

Java Platform, Enterprise Edition (Java EE) applications and modules include an Extensible Markup Language (XML)-based deployment descriptor that specifies various Java EE artifacts that pertain to applications or modules. The Java EE 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 Java EE specification requires that these artifacts map to Java EE platform-specific information, such as that found in WebSphere Application Server, during deployment of Java EE 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 Java EE 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.
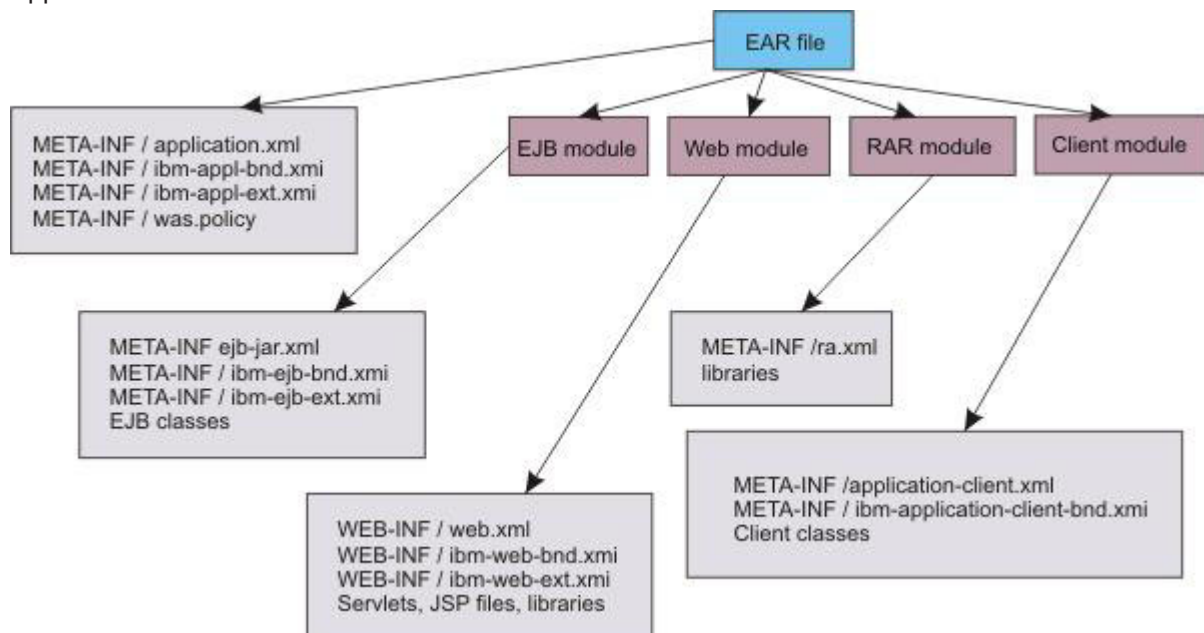
The application management architecture provides a set of classes with which deployers can collect WebSphere Application Serverdeployment information. This information is also referred to as *binding*

information, and is stored in the application EAR file. The deployer can install the EAR file into aWebSphere Application Serverconfiguration 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 Java EE applications as part of product installation.

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 install your application.

About this task

Perform the following tasks to start an application through programming.

Procedure

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 the product.
// 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)
```

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.

Procedure

1. Create a configuration service proxy object.
2. Create a session.
3. Pass the session information to the AppManagement MBean.

   Every method on the AppMangement 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 440.

```
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.

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.

Procedure
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();
 }
 }
 }
}
```

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 install the application.

About this task

Perform the following tasks to edit your deployed application.

Procedure

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();
        }

    }

}
```

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 install your application.

About this task

Perform the following tasks to completely update an application through programming.

Procedure

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. After you update the application, 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";
```

Windows

```
String fileContents = "C:\test\test.ear";
```

Linux   **HP-UX**   **Solaris**   **AIX**

```
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 compressed .zip file.

Before you begin

To learn about the structure of the compressed .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 install the application.

About this task

Perform the following tasks to add to, update, or delete part of an application through programming.

Procedure

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.

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.
```

Windows

```
String partialApp = "C:\apps\partial.zip";
```

Linux   HP-UX   Solaris   AIX

```
String partialApp = "/apps/partial.zip";

String appName = "MyApp";

//Do a get of the administrative client to connect to the product.

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);
        }
    }
}
```

What to do next

After you update the application, remove the listener and quit.

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 your application.

About this task

Perform the following tasks to add a module to an application through programming.

Procedure

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, 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. 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.

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 specfies 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.
```

Windows

```
String moduleName = "C:\apps\foo.jar";
```

Linux   **HP-UX**   **Solaris**   **AIX**

```
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 the product. 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 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);
       }
    }
}
```

What to do next

After you add the module successfully, remove the listener and quit.

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.

Procedure

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 specfies 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.
```

Windows

```
String moduleName = "C:\apps\foo.jar";
```

```
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);
// When a module is update, there are three possible merge actions:
// AppConstants.APPREDEPL_DEFAULT_MERGE (the default merge action),
//    This option specifies during the update action, the binding information from
//    the new version of the EAR file or module is preferred
//    over the corresponding binding information from the old version. If any element
//    of binding is missing in the new version, the corresponding
//    element from the old version is used.
// AppConstants.APPREDEPL_IGNORE_OLDBND,
//    This option specifies that during the update action, the binding information
//    from the new version of the module is preferred over
//    the corresponding binding information from the old version. The bindings from
//    the old version of the application or module are ignored.
// AppConstants.APPREDEPL_IGNORE_NEWBND
//    This option specifies that during the update action, binding information from
//    the old version of the module is preferred over
//    the corresponding binding information from the new version. If any element of
//    the binding does not exist in the old version,
//    the element from the new version is used.
//
// To find the matching configuration object to perform the merge action, the values
// of the read only fields of the task for the new module are compared
// with the values of the read only fields of the task from the existing module.
// If all the read only values match, then the appropriate merge action is
// performed.
preferences.put (AppConstants.APPREDEPL_IGNORE_NEWBND, Boolean.TRUE);
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 the product. 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 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 install your
application.

About this task

Perform the following tasks to add a file to an application through programming.

Procedure

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. After you add a file, 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";


  Windows
  String fileContents = "C:\temp\abc.jsp";


  Linux    HP-UX    Solaris    AIX
  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 install the application.

About this task

Perform the following tasks to update a file through programming.

Procedure

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 completes, 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.
```

Windows
```
String fileContents = "C:\apps\test.jsp";
```

Linux   **HP-UX**   **Solaris**   **AIX**
```
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);
        }
    }
}
```

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.

Procedure
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 completes, 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);
        }
    }
}
```

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 install the application.

About this task

Perform the following tasks to delete a module through programming.

Procedure

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);
          }
      }
}
```

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 install the application.

About this task

Perform the following tasks to delete a file through programming.

Procedure

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);
        }
    }
}
```

Chapter 11. 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.

Procedure

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 49. Extensible DeploymentConstants operations.   Select an operation to extend..

Operation	Description
DeploymentConstants.CDF_OP_INSTALLJ2EE	Installs a Java Platform, Enterprise Edition (Java EE) EAR file
DeploymentConstants.CDF_OP_EDITJ2EE	Edits a deployment application configuration

Table 49. Extensible DeploymentConstants operations (continued).   Select an operation to extend..

Operation	Description
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 utlity 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 12. 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

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.

Procedure

- Import assets to a repository.
- View, delete, update, or export assets.
- Create a business-level application.
- Create a Service Component Architecture (SCA) business-level application that has SCA assets, shared libraries, or business-level applications.
- Start the application.
- Stop the application.
- Update the application and its configuration units.
- Update SCA composite artifacts.
- View the composite definition of an SCA asset composition unit.
- View SCA domain information.
- View or edit JMS bindings on references and services of SCA composites
- 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:
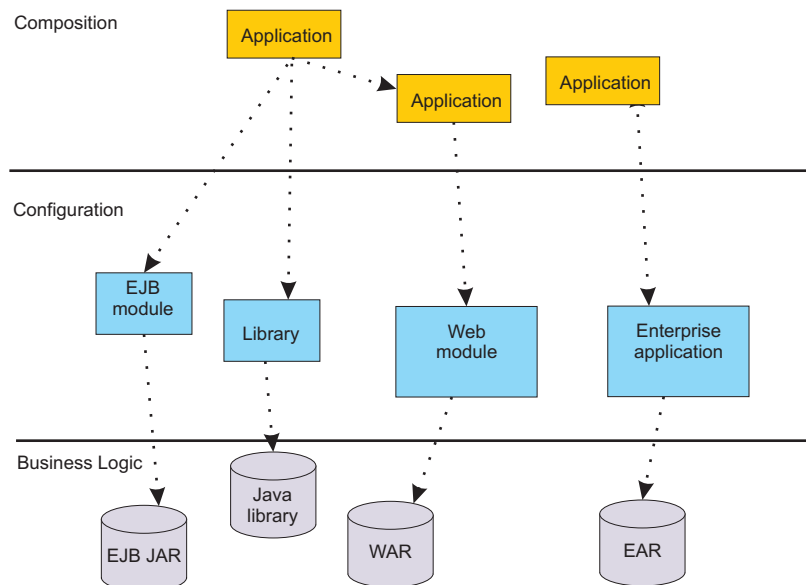
A business-level application does not introduce new programming, runtime, or packaging models:
- You do not need to change your application business logic. The business-level application function does not introduce new application programming interfaces (APIs).
- You do not need to change your application 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 *profile_root*/`config`/ `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 50. Java EE concepts compared to business-level application concepts. Business-level application concepts include assets, composition units, and deployable units.

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: 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: 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).

Table 50. Java EE concepts compared to business-level application concepts (continued). Business-level application concepts include assets, composition units, and deployable units.

Java EE concept	Business-level application concept	Description
Uninstall Java EE application	Multiple business-level application management commands	You delete the Java EE application composition unit from the business-level application: 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.
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.
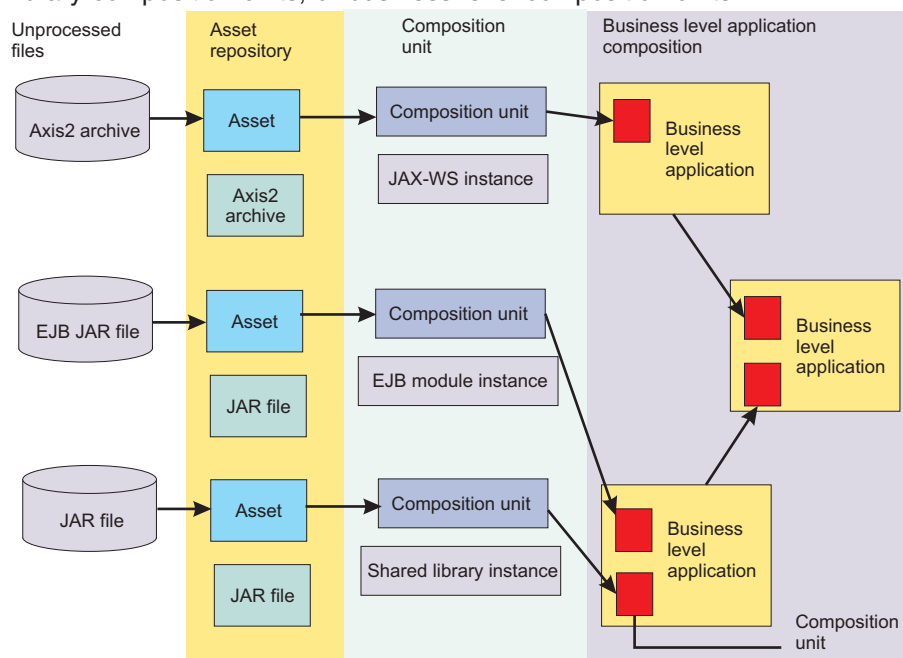
Importing assets

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.

Procedure

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 page, 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 page online help.

      Restriction: OSGi applications do not use a **File permissions** setting.

   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.

      Restriction: OSGi applications do not use a **Validate asset** setting.

   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 page 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 page, 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. If the asset is an OSGi application, additional information about bundle download status is displayed.

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 page 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:

Table 51. Characters that you cannot use in a name. The product does not support these characters in a name.

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.

Information	Value
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.

Information	Value
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, enterprise application archive (EAR) files, and enterprise bundle archive (EBA) files.

The asset type suggests the content of the asset. An asset packaged as a JAR file might contain a web module, portlet, or web service. An asset packaged as an EBA file contains an OSGi application.

This setting is read-only. You cannot edit this setting.

Information	Value
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.

Restriction: OSGi applications do not use a **File permissions** setting.

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.

Table 52. File permission string sets for list options.   Select a list option or specify a file permission string in the text field.

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 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.

Tip: 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:

Table 53. Example URIs for file permission operations.   Results are shown following this table.

Number	Example URL
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

Important: Regardless of the operation system, always use a forward slash (/) as a file path separator in file patterns.

[Windows] You cannot unset read permission on a file on Windows operating systems. With POSIX style permission bits, the bit for denoting `readable` on a file is 4, `writable` is 2, and `executable` is 1. Thus, permission of a file on a Windows operating system is either 5 or 7. Also, in POSIX style there are `user`, `group` and `world` permissions. You can only set the `user` permission for a file on Windows operating systems. The `group` and `world` permission bits are ignored.

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.

Information	Value
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 page:
1. Click **Manage Relationships**. 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**.

Information	Value
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.

Restriction: OSGi applications do not use a **Validate asset** setting.

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.

Information	Value
Data type	String
Default	false (empty check box)

EBA Dependencies

For an enterprise bundle archive (EBA) asset, displays the current bundle download status for all bundles in the asset. This item is only displayed if your asset is an EBA asset, which means that it contains an OSGi application.

You cannot update an EBA asset until bundle downloads are complete from any previous update, and until the business-level application that uses the asset has picked up the previous updates by being restarted. Before you try and update bundle versions, you can use the EBA dependency information to check the bundle download status of the asset. The status displayed is one of the following values:

- Bundles downloading...
- Bundle downloads are complete.
- No bundles downloads are required.

Note: In addition to the information given here, you can also check the bundle download status indirectly, by checking the status of the associated EBA composition unit as described in Checking and updating the EBA asset version used by a business-level application.

If bundle downloads for the asset are complete, or no bundle downloads are required, you can update the asset using either of the methods described in Maintaining bundle versions for an EBA asset.

If bundle downloads for the asset are complete, and a new version of the EBA asset is available, restart the business-level application to bring the EBA composition unit up-to-date and to run the newer configuration.

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.

Procedure

- 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:

Table 54. Button descriptions. Use the buttons to manage assets.

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.
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.

Table 54. Button descriptions (continued).   Use the buttons to manage assets.

Button	Resulting action
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 use the Update asset wizard to update classes, composites, wsdl, xsd, and `definitions.xml` files in an asset.

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 or part of the contents of assets that are in the product management domain. Complete the steps in the Procedure to update an asset using the administrative console Update asset wizard. Alternatively, you can update assets using programming or the wsadmin tool.

The following update limitations exist if the asset you are updating is a Service Component Architecture (SCA) asset:
- You cannot delete a composite file that a composition unit is using. If a delete is attempted, a warning message is sent to the Update asset log.
- You cannot update an `sca-contribution.xml` file.
  - SCA cannot detect deployable composites that are either added or deleted. Therefore, during deployment of a new composition unit, you do not see the new deployable composite in the deployables option list.
  - SCA cannot detect dependencies that are added/removed during the Update asset process.
    - If a new import package is added and if a class in an existing composition unit is updated to require this new package, then the Update asset wizard fails with a ClassNotFoundException. Deployment of any new composition units from the updated asset are successful as the dependencies are detected during deployment operation.
    - If a new export package/namespace is added, then it has no affect on the existing composition unit and the Update asset wizard completes successfully.
- You cannot update configuration metadata such as composite, XSD, or WSDL files in a dependent SCA asset that is deployed as a shared library. You can only update binary files that do not contain annotations which the SCA programming model uses or depends on.
- Because the Update asset wizard uses the new composite definition file provided in the asset for the existing composition unit, the following post deployment related changes to the composite configuration are not saved.
  - Binding resources: If you want to save this information, export all the data to the composite definition file in the new asset before you do the update.
  - Component reference target URIs: If you want to save this information, export all the data to the composite definition file in the new asset before you do the update.

- Component properties: If you want to save this information, export all the data to the composite definition file in the new asset before you do the update.
- HTTP Endpoint URL information: You need to reconfigure this information after the Update asset wizard finishes.
- For web services policy set attachments, during Update asset processing:
  - If there is a policy set specified for an endpoint in the updated composite definition file, SCA checks to see if a policy set has already been attached to that endpoint in the deployed composition unit. If an attachment already exists for that endpoint, the attachment is removed, and the policy set listed in the new composite file for that endpoint is attached. In this situation, if you have made any post deployment policy set configuration changes, these changes are lost.
  - If there is no policy set defined for an endpoint in the update composite definition file, then any existing attachments to that endpoint are removed.

  Policy set bindings follow these same rules.
- For RunAs and RoleToUser mapping definitions, during Update asset processing:
  - For implementation.java, implementation.spring and implementation.osgiapp, any new roles defined in the `definition.xml` file in the asset are picked up and users can be mapped to these roles using either the editCompositionUnit command or the administrative console. Any existing role mappings for the original roles are preserved.
  - For implementation.jee, the runAs and RoleToUser mappings are defined in the JEE application instead of in the SCA asset or SCA composition unit. Therefore, SCA does not do anything with these mappings during Update asset processing.
- The user defined virtual host that hosts web content for binding.ws, binding.atom, binding.http with wireformat.jsonrpc and implementation.widget is not supported. A virtual host mapping of `default_host` is used during Update asset processing.

Procedure

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 page, 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 page 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 page 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 page 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 page, 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.

Update associated composition unit:

Specifies whether to update the composition units that are associated with an enterprise (Java EE) asset. This option applies to enterprise assets only.

The default value is `NONE`. Specify `ALL` to update all of the composition units that are associated with the enterprise asset.

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.

Procedure

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.

Procedure

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 or later 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.

Procedure

1. Select a way to create your business level application.

Table 55. Ways to create business level applications. *You can create business-level applications using the administrative console, programming, or wsadmin.*

Option	Method
Administrative console business-level application creation wizard See "Creating business-level applications with the console" on page 485.	Click **Applications** > **New application** > **New Business-level Application** and follow instructions in the wizard.

Table 55. Ways to create business level applications (continued). You can create business-level applications using the administrative console, programming, or wsadmin.

Option	Method
Administrative console Java Platform, Enterprise Edition (Java EE) application installation wizard See "Installing enterprise application files with the console" on page 278.	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 page.

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.

Procedure

1. Create an empty business-level application.
   a. Click **Applications** > **New application** > **New Business Level Application**.
   b. On the New business-level application page, 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 page, 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 page, change the composition unit settings as needed, and then click **Next**.

This page 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 page, change the deployment target as needed, and then click **Next**.

This page 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 "Relationship options settings" on page 494.

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 page, specify the Java EE application or module to install and click **Next**.

c. On the second Preparing for the application installation page, 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 page 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 page 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 page 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, you can specify composition units for relationships to shared libraries that are used by your business-level application. Specify relationships to shared libraries on the Map shared library relationship page 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 page, 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 page of the wizard, select your business-level application for **Business-level application name**.

f. On the Map shared library relationship page 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:

Table 56. Button descriptions. Use the buttons to manage business-level applications.

Button	Resulting action
Start	Attempts to run the application. After the application starts successfully, the state of the application changes to *Started* if the application starts on all deployment targets, else the state changes to *Partial Start*.
Stop	Attempts to stop the processing of the application. After the application stops successfully, the state of the application changes to *Stopped* if the application stops on all deployment targets, else the state changes to *Partial Stop*.
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.

Table 57. Application status. The status indicates whether the application is running.

Icon	Application status	Description
	Started	Application is running.
	Partial start	Application is in the process of changing from a *Stopped* state to a *Started* state. Application is starting to run but is not fully running yet. Or, it cannot fully start because 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 *Started* state to a *Stopped* state. Application has not stopped running yet.
	Unknown	Status cannot be determined. On single-server installations, 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**.

New business-level application settings

Use this page to name and describe a new business-level application.

To view this administrative console page, 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:

Table 58. Characters that you cannot use in a name. The product does not support these characters in a name.

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	

Information	Value
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 page of the application installation wizard.

To view this console page 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 to 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 identifers:

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 page 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.

Note: If you later change the deployment target of the business-level application or its modules, you must manually update the shared library target to match the target of the application and modules. The targets of shared library composition units are not automatically updated. Not updating the target of the shared library composition unit might cause java.lang.ClassNotFoundException errors and prevent the application or its modules from starting. To prevent these error conditions, also ensure that shared libraries upon which other modules or applications depend have a lower starting weight than dependent applications and modules.

Add composition unit settings

Use this page 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 page, 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 page to add one or more assets to a business-level application.

To view this administrative console page, 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 page 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 page, 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_type*`name=`*unit_name*`,`*unit_type*`version=`*version_number*. For example, for the `MyApp.jar` asset, the backing identifier might be `WebSphere:assetname=MyApp.jar`.

Information	Value
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:

Table 59. Characters that you cannot use in a name.   The product does not support these characters in a name.

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	

Information	Value
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.

Information	Value
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.

Information	Value
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.

Table 60. Restart behavior on update options.   Depending on your selection, the product restarts all target nodes, the nodes controlled by sync plug-ins, or no nodes.

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 page 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 page, 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 page in the add composition unit wizard.

On single-server products, a deployment target can be an application server or web server.

On this page, 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 8.5 application programming interface (API) or uses an 8.5 feature, then you must map the unit to an 8.5 deployment target. If the unit supports Java Platform, Enterprise Edition (Java EE) 6, then you must map the unit to an 8.0 or later deployment target.

If the unit calls a Version 7.x API, uses a 7.x feature, or supports Java EE 5, then you must map the unit to an 8.x or 7.x deployment target.

If the unit supports Java 2 Platform, Enterprise Edition (J2EE) 1.4, then you must map the unit to an 8.x, 7.x or 6.x deployment target. You can map units that call a 6.x API or use a 6.x feature to an 8.x, 7.x or 6.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 **<<**.

Relationship options settings

Use this page to specify relationship options for deployable or composition units in an asset deployed as part of a business-level application. Specifying a relationship declares a dependency relationship that a deployable unit or composition unit has on another asset deployed as a shared library in the same business-level application.

To view this administrative console page, click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_name* > **Relationship options**. This help also pertains to wizard pages that are shown when you add multiple deployable or composition unit assets to a business-level application. These pages 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 page 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 page 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 page 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 page.
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.
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 or composition unit name:

Specifies the name of the deployable unit or the composition 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 page.

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:

Table 61. Characters that you cannot use in a name. The product does not support these characters in a name.

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

Table 61. Characters that you cannot use in a name (continued). The product does not support these characters in a name.

Unsupported characters		
; semi-colon	**#** hash mark	**&** ampersand (and sign)
? question mark	**]]>** No specific name exists for this character combination	

Information	Value
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.

Table 62. Deployed assets button descriptions. Use the buttons to add or delete composition units.

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 page 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 page, 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.

Table 63. Business-level applications button descriptions. Use the buttons to add or delete composition units.

Button	Resulting action
Add	Opens a wizard that helps you add a business-level application to your business-level application.

Table 63. Business-level applications button descriptions (continued). Use the buttons to add or delete composition units.

Button	Resulting action
Delete	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. On single-server installations, deletion occurs after the configuration is saved.

Composition unit settings

Use this page to view composition unit settings and to change the configuration properties of a composition unit. The specific settings that are available for configuration can vary, depending upon the contents of the composition unit. For example, there are additional configuration settings if the asset contained in the composition unit is an SCA composite, or an OSGi application.

To view this administrative console page, click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_name*. The deployed asset is a composition unit of the business-level application.

- "Settings that are common to all composition units"
- "Additional composition unit settings for SCA composites" on page 499
- "Additional composition unit settings for OSGi applications" on page 499

Settings that are common to all composition units

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.

Backing ID:

Specifies a unique identifier for a composition unit that is registered in the application management domain.

The identifier has the format `WebSphere:`*unit_type*`name=`*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.

Information	Value
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.

Note: Assign composition units upon which other composition units depend a lower starting weight than the dependent composition units. If a composition unit is not started and running before its dependent composition units, java.lang.ClassNotFoundException errors might result when you attempt to start the application or its modules.

Information	Value
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.

Information	Value
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.

Table 64. Option descriptions. Specifies whether to restart an asset or shared library composition unit.

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

Target mapping:

Specifies the current targets for the composition unit.

To change the deployment targets, click **Modify targets** then select a different set of deployment targets from the list of available clusters and servers.

For SCA, you must specify only a single server or cluster as the target. Do not map an SCA composition unit to multiple servers or clusters.

Note: When you change the deployment target of composition units in a business-level application, the startup order changes to the same order in which you remap composition unit targets, even if the starting weight for all composition units is set to 1. To avoid java.lang.ClassNotFoundException errors when attempting to start the remapped composition units, remap targets for composition units in the same order as that used to add the composition units or, after remapping, check starting weights to ensure that composition units upon which other composition units depend are started first.

Additional composition unit settings for SCA composites

SCA composite components:

Specifies the component names and component implementations of SCA composites in the application.

Table 65. Column descriptions. Provides the name of each component and the name of the class or code implementing the component.

Column	Description
Component Name	Specifies the name of a component associated with the SCA composite.
Component Implementation	Specifies the name of the class or code implementing the component.

None indicates that the SCA composite does not have defined components.

SCA composite properties:

Specifies the names and values of SCA composite properties in the application.

Table 66. Column descriptions. Provides the name and value of SCA composite properties.

Column	Description
Property Name	Specifies the name of an SCA composite property.
Property Value	Specifies the value of the property.

None indicates that the SCA composite does not have defined name-value properties.

SCA composite wires:

Specifies the sources and targets of wires in the SCA composite.

Table 67. Column descriptions. Provides the source and target of wires.

Column	Description
Wire Source	Specifies the source of a wire in the SCA composite.
Wire Target	Specifies the target of the wire.

None indicates that the SCA composite does not have defined wires.

Additional composition unit settings for OSGi applications

OSGi application deployment status:

The deployment status shows whether updates are available for the EBA asset that is contained in the composition unit. If a new version of an EBA asset is available, and all bundle downloads for the asset are complete, you can update the EBA composition unit so that the business-level application uses the latest configuration. You do not have to update the composition unit every time you update the asset.

There are four distinct deployment statuses for an EBA composition unit:

Using latest OSGi application deployment.
> The composition unit is running the latest configuration of the backing asset and any CBA extensions.

New OSGi application deployment not yet available because it requires bundles that are still downloading.
> The backing asset is currently undergoing a bundle version update, or bundles are downloading for a CBA extension.

New OSGi application deployment available.
> The backing asset is available at a newer configuration than the configuration that is currently running in this composition unit, or a CBA extension has been added or replaced.

New OSGi application deployment cannot be applied because bundle downloads have failed.
> The last bundle version update for the backing asset or CBA extension did not succeed, and therefore the newer configuration is not yet available.

If the status is "New OSGi application deployment available", the **Update to latest deployment ...** button is available. Click this button to bring the EBA composition unit up-to-date and run the updated business-level application. If any of the updates need configuration changes, a wizard prompts you to update the configuration information.

When you save the changes to the EBA composition unit, the associated business-level application is updated to use the new configuration. If the business-level application is running, the bundle and configuration updates are applied immediately. If possible (that is, depending on the nature of the updates) the system applies the updates without restarting the application. Updates that pull in new use bundles at run time prompt a full restart of the application. Updates that pull in new provision bundles might also prompt a full application restart.

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 compressed 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.

Procedure

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 compressed file and click **Next**.

   c. On the Select options for importing an asset page, click **Next**.

   d. On the Summary page, click **Finish**.

   e. On the Adding asset to repository page, if messages show that the operation completed, click **Manage assets**.

f. 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 page, 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 page, 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 page, click **Next**.

   e. On the Map composition unit to a target page, change the target server as needed, and then click **Next**.

   f. On the Summary page, 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.

SCA application package deployment

The product supports deployment of many types of Service Component Architecture (SCA) artifacts as composition units of business-level applications. Typical artifacts include Java archive (JAR) files, compressed `.zip` files, and web application archive (WAR) files.

The following outlines the details about deployment of SCA artifacts:
- Deployment of JAR or compressed files
- Deployment of WAR files
- Notes and limitations

Deployment of JAR or compressed files

- The product supports one composite file for each package. The product determines which composite file to support using the following process:

1. The SCA deployment extension looks for the `META-INF/sca-contribution.xml` file, gets the name of each deployable composite defined in the file, and uses QName values to find the actual composite file names under any directory for that composite. If more than one composite is found in the `sca-contribution.xml` file, you can select the composite to deploy.

2. If there is no `META-INF/sca-contribution.xml` file defined, the SCA deployment extension looks for a composite file in the `META-INF/sca-deployables` directory.

- The product validates SCA composites for inconsistencies with SCA specifications.

  One specification requirement is that the names of top-level components must be unique. Thus, the product validates top-level component name uniqueness.

  Tip: Top-level components are also called domain components, with the top-level or domain typically the cell on multiple-server installations and the server scope on single-server installations.

  The product validates all composite files in a JAR or compressed file, regardless of the file location in the archive or whether the `sca-contribution.xml` file references the composite file. The product does not validate all services and references.

  The product writes warning and error messages resulting from the validation tests to the `SystemOut.log` file. Refer to the log file to learn about inconsistencies with specifications in your SCA composites.

  Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

- The product uses the following process for QName resolution:

  – The product uses the QName to resolve composite files included in the top-level composite that use the element. For example, the `<include name="mynamespace:MyService"/>` statement looks for a composite file whose composite name is `MyService` and whose targetNameSpace is `mynamespace`. The following rules apply:
    - **name**: Use the outer composite.
    - **namespace declarations**: Merged into the outer composite.
    - **targetNamespace**: Use the outer composite (must be the same).
    - **local**: Use the composite (preferably the same but not required).
    - **requires(intents) and policySets**: Must be compatible, and aggregated into the outer composite.

    Deployable composite files must have name and targetNamespace values. The name and targetNamespace values are combined to form the QName of a composite file.

  – When a composite is used as a component implementation in the top-level composite, the composite is also resolved using the QName. For example, the `<implementation.composite name="mynamespace:MyComposite"/>` statement causes the product administration to look for a composite file whose composite name is `MyComposite` and whose targetNamespace is `mynamespace`.

- A JAR file can contain other JAR files at the top level. The contained JAR files are available on the classpath. However, any archives inside those JAR files are not available. The product supports one level of embedded JAR files.

Deployment of WAR files

- A composite file in a WAR file must be named `default.composite`. A composite file that is not in a WAR file can have any name.

- The `default.composite` composite file must be inside a WAR file in the `META-INF/sca-deployables` directory.

- The `META-INF/sca-deployables` directory must contain no more than one composite file. If there is more than one composite file in the `META-INF/sca-deployables` directory, then the product returns a validation error and stops the WAR file deployment.

  However, you can place other composite files in directories other than `META-INF/sca-deployables`, and reference those composite files in the top-level composite under the `META-INF/sca-deployables` directory.

- The product does not support having a `sca-contribution.xml` file inside the WAR file under the `META-INF` directory. If the product finds a `sca-contribution.xml` file, then the product returns a validation error and stops the WAR file deployment.

Notes and limitations

- The product does not provide administration console pages for configuring contributions.

- If you import a package or namepace from a different contribution, or JAR (`contribution.xml`), you might need to import that contribution as an asset before importing your own asset.

  For example, suppose your Contribution A imports a JAR file from Contribution B. You might need to import Contribution B and then Contribution A as assets. Contribution A depends on Contribution B so you must import Contribution B before importing Contribution A.

- You cannot use a local interface across a class loader boundary. Use a remotable interface to cross a class loader boundary.

Creating SCA business-level applications

You can create an empty business-level application and then add Service Component Architecture (SCA) assets, shared libraries, business-level applications, and other artifacts as composition units to the empty business-level application.

Before you begin

Configure the target application server. You must deploy SCA composite assets of a business-level application to a Version 8.x server (target) or to a Version 7.0 target that is enabled for the Feature Pack for SCA.

If your SCA composite or application uses OASIS support, you must deploy the SCA asset or application to a Version 8.5 target.

If your SCA composite or application uses Feature Pack for SCA Version 1.0.1 functionality, you must deploy the SCA asset or application to a Version 8.x target or to a feature pack Version 1.0.1.0 target. Version 1.0.1 functionality includes:
- Java Message Service (JMS) bindings
- Atom bindings
- HTTP bindings with a wire format of JSON-RPC
- Java Platform, Enterprise Edition (Java EE) integration modules (`implementation.jee`, `implementation.web`, or `implementation.ejb` components)
- SCA Spring component implementations
- OSGi applications as SCA component implementations
- Service Data Objects (SDO) composites

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, the wsadmin tool, or programming.

You create SCA business-level applications the same way as for non-SCA business-level applications. However, when you use an SCA asset in a business-level application, function that applies only to applications that use SCA composites becomes available. For example, you can access administrative console pages that apply only to applications that use SCA composites.

Procedure

1. Select a way to create your business-level application.

Table 68. Ways to create SCA business-level applications.   You can create a business-level application using the administrative console, wsadmin scripts, or programming.

Option	Method
Administrative console business-level application creation wizard See "Creating SCA business-level applications with the console" on page 505.	Click **Applications** > **New Application** > **New Business Level Application** and follow instructions in the wizard. For example use of the console to create a business-level application that has an SCA asset, see "Example: Creating an SCA business-level application with the console" on page 535.

2. Create your business-level application using the administrative console, wsadmin, or programming.
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 SCA or other assets needed by your business-level application.
2. Add assets, shared libraries, or other business-level applications as composition units.

   If the asset or application uses OASIS support, specify a Version 8.5 target.

   If the asset or application uses Feature Pack for SCA Version 1.0.1 functionality, specify a Version 8.x target or a feature pack Version 1.0.1.0 target.

   For applications that use `implementation.osgiapp`, add the enterprise bundle archive (EBA) asset as a composition unit to the business-level application before adding the SCA asset as a composition unit.
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.

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. If SCA composite assets do not start, ensure that each asset is mapped to a deployment target that supports SCA composites.

When an SCA application fails to start, multiple first failure data capture (FFDC) entries are logged for a single error. The FFDC log entries pertain to the same problem and are not different issues related to the failure. Use the information provided in the FFDC log entries to fix the problem and try starting the SCA application again.

If an asset composition unit uses an Enterprise JavaBeans (EJB) binding and does not start because it has a `non-WebSphere target of "null"`, delete the asset composition unit and add it again to the business-level application. Specify a target that supports SCA composites when you add the asset to the business-level application. You cannot change the target after deployment.

If the `META-INF/sca-deployables` directory has multiple SCA composite files and the application does not start because the product `cannot obtain the CompUnitInfoLoader value`, place only the file that contains the composite in the `META-INF/sca-deployables` directory. You can place the other composite files anywhere else within the archive.

If the SCA application uses security, the target must be in the global security domain.

For applications that use `implementation.osgiapp` in multiple-node environments, target the EBA composition unit to the same server or cluster as the SCA composition unit.

Creating SCA business-level applications with the console

You can create an empty business-level application and then add Service Component Architecture (SCA) assets, shared libraries, or business-level applications as composition units to the empty business-level application.

Before you begin

Configure the target application server. You must deploy SCA composite assets of a business-level application to a Version 8.x server (target) or to a Version 7.0 target that is enabled for the Feature Pack for SCA.

If your SCA composite or application uses Feature Pack for SCA Version 1.0.1 functionality, you must deploy the SCA asset or application to a Version 8.5 target or to a feature pack Version 1.0.1.0 target. Version 1.0.1 functionality includes:
- Java Message Service (JMS) bindings
- Atom bindings
- HTTP bindings with a wire format of JSON-RPC
- Java Platform, Enterprise Edition (Java EE) integration modules (`implementation.jee`, `implementation.web`, or `implementation.ejb` components)
- SCA Spring component implementations
- OSGi applications as SCA component implementations
- Service Data Objects (SDO) composites

Also, determine an application name. Optionally, determine which assets, shared libraries, or business-level applications that the new business-level application needs.

About this task

You can create a business-level application that has SCA assets using the administrative console. Alternatively, you can use the wsadmin scripting tool or programming.

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.

Procedure

1. Create an empty business-level application.
   a. Click **Applications** > **New Application** > **New Business-level Application**.
   b. On the New business-level application page, specify a unique name for the application and a description, and then click **Apply**.
   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 `Unknown`.

2. Add one SCA asset to your business-level application. The product adds the asset as a composition unit of your business-level application.

   a. Import the SCA asset.

   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.

      Although you can add an asset, shared library, or business-level application to your business-level application, the logic is in your SCA asset. Add the SCA asset as a composition unit.

      Under **Deployed assets**, click **Add** > **Add Asset**.

   d. On the Add page, select one unit from the list of available units, and then click **Continue**.

      On the Add page, you might be able to select multiple deployable SCA composites. However, you can deploy only one deployable SCA composite at a time. Select only one unit and click **Continue**. If you select multiple units, the product deploys only one of those units.

      For applications that use `implementation.osgiapp`, add the enterprise bundle archive (EBA) asset as a composition unit to the business-level application before adding the SCA asset as a composition unit.

   e. On the Set options page, change the composition unit settings as needed, and then click **Next**.

      This page is not shown if you have multiple deployable unit assets.

   f. On the Map composition unit to a target page, specify one target server that supports SCA composites, and then click **Next**.

      The target server can be an existing cluster. To map the composition unit to a cluster, select the existing cluster from the **Available** list, click **Add**, and then click **Next**. The cluster name is shown in the **Current targets** list as `WebSphere:cluster=cluster_name`.

      If you are adding an SCA asset that uses security, specify a target server that is in the global security domain.

      For applications that use `implementation.osgiapp` in multiple-node environments, target the EBA composition unit to the same server as the SCA composition unit.

      This page is not shown when you add a business-level application.

   g. On the Relationship options page, click **Next** to accept the default values.

      The relationships in SCA applications are set at the asset level. Either the asset must be defined as an SCA contribution or, in the asset view, a relationship must be set to another asset. When a relationship is set to another asset manually at the asset level, the relationship only exposes all the packages within the asset to the other depended asset. The namespaces are not exposed.

      This page is shown only for SCA assets that have multiple deployable or composition units.

   h. On the Set Java EE composition unit relationship page, associate SCA components with Java EE applications and then click **Next**.

      Java EE applications are also known as enterprise applications or enterprise archive (EAR) files. An SCA composite definition can specify an EAR file to use on the `archive` attribute of an `implementation.jee` tag. Use this page to associate SCA components in this business-level application to the EAR files named in the composite definition. If your SCA application does not use EAR files, take the default values and click **Next**.

   i. On the Map security roles to users or groups page, specify security roles for users or groups as needed, and then click **Next**.

      This page is only shown for SCA assets that use security.

   j. On the Map RunAs roles to users page, map a user identity and password to RunAs roles as needed, and then click **Next**.

      This page is only shown for SCA assets that use security.

   k. On the Map virtual host page, specify a virtual host that hosts web services for each SCA composite, and then click **Next**. By default, composites map to `default_host`.

This page is only shown for SCA assets that contain a web service binding.

l. On the Attach policy set page, attach a policy set and assign policy set bindings as needed, and then click **Next**.

This page is only shown for SCA assets that use web services.

m. 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*`]` indicates that the addition is successful. Click **Manage application**.

If the product adds the unit successfully, the name of the unit is shown in a list of deployed assets on the business-level application settings page.

If the unit addition is not successful, read the messages and add the unit again. Correct the problems noted in the messages.

n. On the Adding composition unit to the business-level application page, click **Save**.

3. Optional: Add one or more assets, shared libraries, or business-level applications to your business-level application.

Repeat Step 2 to add another asset or add a shared library or business-level application.

Results

A business-level application that contains the specified composition units.

What to do next

After you create the application, save the changes to your configuration and start the application as needed.

If a composite asset is deployed to a target that does not support SCA composites, the SCA composite does not start. You must deploy an SCA asset to a target that supports SCA composites.

Map virtual host settings for SCA composites

Use this page to map Service Component Architecture (SCA) composites that use a web service binding to a virtual host. You must map the composites to the virtual host that hosts the web services.

This administrative console page displays in the business-level application creation and update wizards. To view the Map virtual host page, the asset that you add to a business-level application must contain a web service binding. To view this page, do the following:

1. Import an asset that contains a web service binding.
2. Create a business-level application to which to add the asset.
3. Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > **Add** > **Add Asset**.
4. On the Add composition unit page, select the asset that contains a web service binding, and click **Continue**.
5. On the Set options page, change the settings as needed and click **Next**.
6. On the Map composition unit to a target page, specify target servers as needed and click **Next**.
7. On the Define relationship with existing composition units page, change the settings as needed and click **Next**.
8. Continue changing settings as needed and click **Next** on any other pages until the Map virtual host page is displayed in the wizard.

Composite Name:

Specifies the name of the composite that uses a web service binding in the SCA artifact.

Virtual Host:

Specifies a virtual host to associate with the composite.

Select the virtual host that hosts the web services for the composite. By default, the product associates a component with the `default_host` virtual host.

Set Java EE composition unit relationships for SCA composites

Use this page to associate Service Component Architecture (SCA) components in an SCA composite with Java Platform, Enterprise Edition (Java EE) applications, otherwise known as *enterprise applications* or *enterprise archive (EAR) files*.

An SCA composite definition can define Java EE applications as component implementations. You can define an EAR asset on an `archive` attribute of the `implementation.jee` tag for the component and use the application deployed from the asset as its implementation. On this page, associate SCA components in a business-level application with the EAR files named in the composite definition.

This administrative console page displays in the business-level application creation and update wizards. To view the Set Java EE composition unit relationship page, the asset that you add to a business-level application must contain an SCA composite. To view this page, complete the following actions:

1. Import an asset that contains an SCA composite.
2. Create a business-level application to which to add the asset.
3. Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > **Add** > **Add Asset**.
4. On the Add composition unit page, select the SCA composite asset and click **Continue**.
5. On the Set options page, change the settings as needed and click **Next**.
6. On the Map composition unit to a target page, specify target servers as needed and click **Next**.
7. On the Define relationship with existing composition units page, change the settings as needed and click **Next**.
8. Continue changing settings as needed and click **Next** on any other pages until the Set Java EE composition unit relationship page is displayed in the wizard.

Component Name:

Specifies the name of an SCA component in the SCA composite that you are deploying.

EAR Asset Name:

Specifies the name of the enterprise application, or EAR file, that the SCA component uses.

An EAR asset is an EAR file that has been imported as an asset. The EAR asset name must match the `archive` attribute on the `implementation.jee` tag in the SCA composite definition.

Associated Java EE Composition Unit:

Specifies the composition unit name of the EAR asset. Select the Java EE composition unit that the SCA component uses. To associate an SCA component with an EAR file, the EAR file must be a composition unit of your SCA business-level application.

If the Java EE composition unit that you want to associate with an SCA component is not in the drop-down list, import the EAR file that is named by the `archive` attribute on the `implementation.jee` tag of the SCA composite definition as an asset. Then, add the EAR asset as a composition unit of this business-level application.

You can use the **Import an asset** and **Add an asset** links on this page to add EAR files as assets and make them composition units of your SCA business-level application.

Attach policy set settings

Use this page to attach a policy set and assign policy set bindings for the composite defined in a Service Component Architecture (SCA) application.

This administrative console page displays in the Create new business-level application wizard. To have the Attach policy set page in the wizard, the SCA component in the asset that you add to a business-level application must use a web service binding, `binding.ws`, and the composite file or annotation must specify the intents or policy sets. To view this page, do the following:

1. Import an asset that uses a web service binding and a composite file or annotation that specifies the intents or policy sets.
2. Create a business-level application to which to add the asset.
3. Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > **Add** > **Add Asset**.
4. On the Add composition unit page, select the asset that uses a web service binding, and click **Continue**.
5. On the Set options page, change the settings as needed and click **Next**.
6. On the Map composition unit to a target page, specify target servers as needed and click **Next**.
7. On the Define relationship with existing composition units page, change the settings as needed and click **Next**.
8. Continue changing settings as needed and click **Next** on any other pages until the Attach policy set page is displayed in the wizard.

To attach or detach a policy set or to assign a policy set binding, do the following:

1. Select a composite, component, service, reference, or binding from **Name**. The **Name** list is nested, indicating parent-child relationships. When you select a parent, the children are automatically selected.
2. Click the desired button.

Table 69. Button descriptions. Use the buttons to attach or detach policy sets and to assign policy set bindings.

Button	Resulting action
Attach	Attaches a policy set to the selected composite, component, service, reference, or binding.
	When the **Include default policy sets** option is not enabled, the options for this button contain user-created policy sets only.
	When the **Include default policy sets** option is not enabled and no user-created policy sets exist, then there are no button options. You can select **Include default policy sets** to display the default policy set options.
	When the **Include default policy sets** option is enabled, the options for this button include both default policy sets and any user-created policy sets.
	To attach a policy set, select a composite, component, service, reference, or binding from **Name** and click **Attach** > *policy_set_option*.
	To close the menu list, click **Attach**.
Detach Policy Set	Detaches a policy set from the selected composite, component, service, reference, or binding.

Table 69. Button descriptions (continued).   Use the buttons to attach or detach policy sets and to assign policy set bindings.

Button	Resulting action
Assign Service Policy Set Binding	Assigns a service policy set binding to the selected composite, component, service, reference, or binding. There are two default options: **Default** specifies to assign the default service policy set binding. **Provider Sample** specifies to assign a policy set binding that is provided with the product to the service. If you are deploying the composition unit to a server or cluster that belongs to a security domain, the list of policy set bindings consists of bindings that have been defined in the security domain to which the composition unit is being deployed.
Assign Reference Policy Set Binding	Assigns a reference policy set binding to the selected composite, component, service, reference, or binding. There are two default options: **Default** specifies to assign the default reference policy set binding. **Client Sample** specifies to assign a policy set binding that is provided with the product to the reference. If you are deploying the composition unit to a server or cluster that belongs to a security domain, the list of policy set bindings consists of bindings that have been defined in the security domain to which the composition unit is being deployed.

Include default policy sets:

Specifies whether to include default policy sets. Default policy sets specify common quality of service (QoS) behavior for generic message format.

Before selecting this option, determine whether the default policy sets provide adequate QoS characteristics for your services.

By default, this option is not enabled.

Name:

Specifies a composite, component, service, reference, or binding in the artifact.

The **Name** list is nested, indicating parent-child relationships. When you select a parent, the children are automatically selected.

Intents:

Specifies the aggregate of the intents from the composite file and the annotations. SCA intents are used to describe the abstract policy requirements of a component.

The intents shown include any intents inherited from a parent.

Matched Policy Sets:

Specifies policy sets that potentially satisfy the intents.

You can include default policy sets by enabling the **Include default policy sets** check box. To exclude default policy sets, deselect the check box.

Attached Policy Set:

Specifies attached policy sets. If no value is shown, then the composite, component, service, reference, or binding is not attached to a policy set.

To attach a policy set, select a composite, component, service, reference, or binding and click an **Attach** option.

To detach a policy set, use **Detach Policy Set**. You can detach any policy set, including pre-attached policy sets.

Policy Set Binding:

Specifies service and reference policy set bindings. If no value is shown, then the composite, component, service, reference, or binding is not assigned to a policy set binding.

To assign a policy set binding, select a composite, component, service, reference, or binding and click an **Assign Service Policy Set Binding** or **Assign Reference Policy Set Binding** option.

To reset the bindings, select the **Default** option. For example, select **Assign Service Policy Set Binding** > **Default** or **Assign Reference Policy Set Binding** > **Default**.

Map security roles to users or groups collection for SCA composites

Use this page to view and manage mappings of security roles to users and groups that are used with the Service Component Architecture (SCA) composites.

To view this administrative console page, click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_composition_unit_name* > **Map security roles to users or groups**. This page is the same as the Map security roles to users or groups page in the Create new business-level application wizard. To view this page, your composition unit must support SCA security.

Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to access applications defined by the role. Users, groups, and roles are defined when an application is installed or configured.

To map a role to a user or group, enable the Select check box beside the role name in the list and click a button. On the displayed page, specify one or more users or groups to map to the role.

Table 70. Button descriptions. Use the buttons to map security roles to users, groups, or special subjects.

Button	Resulting action
Map Users	Displays the Map users or groups page on which you can specify the users to have the selected security role.
Map Groups	Displays the Map users or groups page on which you can specify the groups to have the selected security role.

Table 70. Button descriptions (continued).   Use the buttons to map security roles to users, groups, or special subjects.

Button	Resulting action
Map Special Subjects	Maps special subjects according to the option that you select: **None** specifies to map none of the special subjects to the role. **All Authenticated in Application's Realm** specifies to map all of the authenticated users to a specified role. When you map all authenticated users to a specified role, all of the valid users in the current registry who have been authenticated can access resources that are protected by this role. **All Authenticated in Trusted Realms** specifies to map all of the authenticated users in the trusted realms to a specified role. This option gives all authenticated users who belong to the user registry access to the application's realm and all authenticated users who belong to user registries access to realms which are trusted by the current security domain. **Everyone** specifies to map everyone to a specified role. When you map everyone to a role, anyone can access the resources that are protected by this role and, essentially, there is no security.

Role:

Specifies a security role.

Special Subjects:

Specifies which special subjects are mapped to the security role. This option applies only when an application uses multiple realms.

None   Specifies to map none of the special subjects to the role.

All Authenticated in Application's Realm
Specifies to map all of the authenticated users to a specified role. When you map all authenticated users to a specified role, all of the valid users in the current registry who have been authenticated can access resources that are protected by this role.

All Authenticated in Trusted Realms
Specifies to map all of the authenticated users in the trusted realms to a specified role. All authenticated users who belong to the user registry that is mapped to the application's realm and all authenticated users who belong to user registries that are mapped to realms which are trusted by the current security domain are successfully authorized.

Everyone
Specifies to map everyone to a specified role. When you map everyone to a role, anyone can access the resources that are protected by this role and, essentially, there is no security.

To change the value, select the role, click **Map Special Subjects**, and select an option.

Users:

Lists the users that are mapped to the specified role within this application.

Users from the non-default realm are displayed as *user@realm*.

Groups:

Lists the groups that are mapped to this specified role within this application.

Map RunAs roles to users collection for SCA composites

Use this page to map a specified user identity and password to a RunAs role for a Service Component Architecture (SCA) composite. This page enables you to specify application-specific privileges for individual users to run specific tasks using another user identity.

To view this administrative console page, click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_composition_unit_name* > **Map RunAs roles to users**. This page is the same as the Map RunAs roles to users page in the Create new business-level application wizard.

To view this page, the components in your composition unit must contain predefined RunAs roles and support SCA security. RunAs roles are used by components that need to run as a particular role for recognition while interacting with another component.

Username:

Specifies a user name for the RunAs role user.

This user already maps to the role specified in the Mapping users and groups to roles page. You can map the user to its appropriate role by either mapping the user to that role directly or mapping a group that contains the user to that role. After you specify the user name and password for the user and select a RunAs role, click **Apply**.

Password:

Specifies the password for the RunAs user.

Role:

Specifies a security role for a user within this application.

The authorization policy is only enforced when security is enabled.

User:

Lists the user that is mapped to the specified role within this application.

Composition unit settings

Use this page to view composition unit settings and to change the configuration properties of a composition unit. The specific settings that are available for configuration can vary, depending upon the contents of the composition unit. For example, there are additional configuration settings if the asset contained in the composition unit is an SCA composite, or an OSGi application.

To view this administrative console page, click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_name*. The deployed asset is a composition unit of the business-level application.

- "Settings that are common to all composition units" on page 497
- "Additional composition unit settings for SCA composites" on page 499
- "Additional composition unit settings for OSGi applications" on page 499

Settings that are common to all composition units

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.

Backing ID:

Specifies a unique identifier for a composition unit that is registered in the application management domain.

The identifier has the format `WebSphere:`*`unit_type`*`name=`*`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.

Information	Value
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.

Note: Assign composition units upon which other composition units depend a lower starting weight than the dependent composition units. If a composition unit is not started and running before its dependent composition units, java.lang.ClassNotFoundException errors might result when you attempt to start the application or its modules.

Information	Value
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.

Information	Value
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.

Table 71. Option descriptions. Specifies whether to restart an asset or shared library composition unit.

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

Target mapping:

Specifies the current targets for the composition unit.

To change the deployment targets, click **Modify targets** then select a different set of deployment targets from the list of available clusters and servers.

For SCA, you must specify only a single server or cluster as the target. Do not map an SCA composition unit to multiple servers or clusters.

Note: When you change the deployment target of composition units in a business-level application, the startup order changes to the same order in which you remap composition unit targets, even if the starting weight for all composition units is set to 1. To avoid java.lang.ClassNotFoundException errors when attempting to start the remapped composition units, remap targets for composition units in the same order as that used to add the composition units or, after remapping, check starting weights to ensure that composition units upon which other composition units depend are started first.

Additional composition unit settings for SCA composites

SCA composite components:

Specifies the component names and component implementations of SCA composites in the application.

Table 72. Column descriptions. Provides the name of each component and the name of the class or code implementing the component.

Column	Description
Component Name	Specifies the name of a component associated with the SCA composite.
Component Implementation	Specifies the name of the class or code implementing the component.

None indicates that the SCA composite does not have defined components.

SCA composite properties:

Specifies the names and values of SCA composite properties in the application.

Table 73. Column descriptions. Provides the name and value of SCA composite properties.

Column	Description
Property Name	Specifies the name of an SCA composite property.
Property Value	Specifies the value of the property.

None indicates that the SCA composite does not have defined name-value properties.

SCA composite wires:

Specifies the sources and targets of wires in the SCA composite.

Table 74. Column descriptions. Provides the source and target of wires.

Column	Description
Wire Source	Specifies the source of a wire in the SCA composite.
Wire Target	Specifies the target of the wire.

None indicates that the SCA composite does not have defined wires.

Additional composition unit settings for OSGi applications

OSGi application deployment status:

The deployment status shows whether updates are available for the EBA asset that is contained in the composition unit. If a new version of an EBA asset is available, and all bundle downloads for the asset are complete, you can update the EBA composition unit so that the business-level application uses the latest configuration. You do not have to update the composition unit every time you update the asset.

There are four distinct deployment statuses for an EBA composition unit:

Using latest OSGi application deployment.
> The composition unit is running the latest configuration of the backing asset and any CBA extensions.

New OSGi application deployment not yet available because it requires bundles that are still downloading.
> The backing asset is currently undergoing a bundle version update, or bundles are downloading for a CBA extension.

New OSGi application deployment available.
> The backing asset is available at a newer configuration than the configuration that is currently running in this composition unit, or a CBA extension has been added or replaced.

New OSGi application deployment cannot be applied because bundle downloads have failed.
> The last bundle version update for the backing asset or CBA extension did not succeed, and therefore the newer configuration is not yet available.

If the status is "New OSGi application deployment available", the **Update to latest deployment ...** button is available. Click this button to bring the EBA composition unit up-to-date and run the updated business-level application. If any of the updates need configuration changes, a wizard prompts you to update the configuration information.

When you save the changes to the EBA composition unit, the associated business-level application is updated to use the new configuration. If the business-level application is running, the bundle and configuration updates are applied immediately. If possible (that is, depending on the nature of the updates) the system applies the updates without restarting the application. Updates that pull in new use bundles at run time prompt a full restart of the application. Updates that pull in new provision bundles might also prompt a full application restart.

JMS binding settings for SCA composites

Use this page to view property settings for a Java Message Service (JMS) binding that connects Service Component Architecture (SCA) composite references or services. Also use this page to edit resources of the JMS binding. The settings shown on this page define a `binding.jms` element.

To view this administrative console page, your composition unit must support a JMS binding reference or service. In the administrative console, do the following:

1. Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_composition_unit_name*.
2. From the composition unit settings page for SCA composites, select to view references or services:
   - For an SCA component reference, click **SCA Composite Components** > *SCA_component_reference_name* > **SCA Component References** > *reference_name* > **Bindings** > **JMS binding**.
   - For an SCA component service, click **SCA Composite Components** > *SCA_component_service_name* > **SCA Component Services** > *service_name* > **Bindings** > **JMS binding**.

You can use a JMS binding to identify existing JMS resources using Java Naming and Directory Interface (JNDI) names.
- General properties (read-only)
- Resources
- Response resources
- Request header properties for a reference (read-only)
- "Response header properties" on page 520 for a service (read-only)

General properties

Specifies property settings that apply to all `binding.jms` elements. The settings pertain to both services and references. You cannot edit the settings.

JMS binding URI

> Specifies a uniform resource identifier (URI) that identifies properties such as the destination, connection factory and activation specification to be used to send or receive the JMS message.

> The URI has the following format:

```
jms: jms_destination?
connectionFactoryName=connection_factory_name &
destinationType={queue|topic} &
deliveryMode=delivery_mode &
timeToLive=time_to_live &
priority=priority &
user_property=user_property_value & ...
```

Correlation schema

> Specifies the correlation scheme used when sending reply or callback messages.

> Valid values are `requestmsgidtocorrelid` (the default), `requestcorrelidtocorrelid`, and `none`.

Initial context factory

> Specifies the name of the initial context factory used to obtain a JNDI initial context.

> This setting is optional. If no factory is specified, the WebSphere Application Server initial context factory is used.

JNDI URL

> Specifies the uniform resource locator (URL) for the JNDI provider.

Request connection

> Specifies a `binding.jms` element that is present in a composite definition file.

> A request is a message that is sent to an SCA service or sent by an SCA reference.

Response connection

Specifies a `binding.jms` element that is present in a composite definition file.

A response is a message received by a reference (that is, a reply from an invoked service) or a message sent by a service in response to a previous request message. A response in SCA is always a reply to a previous request.

Request wire format

Specifies the component definition element name of a wire that connects SCA composites.

For example, `wireFormat.jmsObject`.

A request is a message that is sent to an SCA service or sent by an SCA reference.

Response wire format

Specifies the component definition element name of a wire that connects SCA composites.

For example, `wireFormat.jmsObject`.

A response is a message received by a reference (that is, a reply from an invoked service) or a message sent by a service in response to a previous request message. A response in SCA is always a reply to a previous request.

Resources

Specifies resources for the JMS binding. You can both read and edit resource settings for **Destination JNDI name**, **Activation specification JNDI name**, and **Connection factory JNDI name**. All other resource settings are read-only.

Destination type

Specifies the type of the request destination. Permitted values are `queue` (the default value) and `topic`. When `topic` is specified, then all the operations in the interface that correspond to the binding must be one-way.

This setting is for both services and references and is read-only.

`@type` is the destination type.

Destination JNDI name

Specifies an optional parameter that gives the JNDI name of a destination to which the binding is connected. For example, `jms/InvokeService_Callback`.

You can set a destination name for both services and references.

`@name` is the JNDI name of the destination.

Destination create

Specifies whether to create a destination for the binding.

Valid values are `ifnotexist` (the default), `always`, and `never`. When the value is `ifnotexist`, the product dynamically creates destination resources necessary for the SCA composite, if those resources do not exist and relate to the default messaging provider. The product creates the resources when adding the SCA composite to a business-level application.

The product validates a composite definition when adding an SCA asset to a business-level application. If the validation results in an error, the product does not add the asset to the application. If the value is `always` and the destination exists, deployment stops with an error.

Activation specification JNDI name

Specifies the activation specification that the binding uses to connect to a JMS destination to process request messages. The value must be a JNDI name.

You can set an activation specification for services only.

The attributes of this element follow those defined for the destination element.

Activation specification create

Specifies whether to create an activation specification for the binding. You can create an activation specification for services only.

Valid values are `ifnotexist` (the default), `always`, and `never`. When the value is `ifnotexist`, the product dynamically creates activation specification resources necessary for the SCA composite, if those resources do not exist and relate to the default messaging provider. If the value is `always` and the activation specification exists when the SCA composite is added to a business-level application, deployment stops with an error.

Connection factory JNDI name

Specifies the connection factory that the binding uses to process request messages. The value must be a JNDI name.

You can set a connection factory JNDI name for references only.

The attributes of this property follow those defined for the destination element. This property is mutually exclusive with the activationSpec property.

Connection factory create

Specifies whether to create a connection factory for the binding. You can create a connection factory for references only.

Valid values are `ifnotexist` (the default), `always`, and `never`. When the value is `ifnotexist`, the product dynamically creates connection factory resources necessary for the SCA composite, if those resources do not exist and relate to the default messaging provider. If the value is `always` and the connection factory exists when the SCA composite is added to a business-level application, deployment stops with an error.

Response resources

Specifies the resources used for handling response messages, receiving responses for a reference, and for sending responses from a service. A response element defines the destination and either the connection factory or activation specification elements for handling response messages.

Response resources pertain to both services and references. You can both read and edit resource settings for **Response destination JNDI name** and **Response connection factory JNDI name**. All other resource settings are read-only.

Response destination type

Specifies the type of the response destination. Permitted values are `queue` (the default value) and `topic`. When `topic` is specified, then all the operations in the interface that corresponds to the binding must be one-way.

This setting is for both services and references and is read-only.

Response destination JNDI name

Specifies the destination that is to be used to process responses by this binding. Attributes are the same as for the parent destination element. For example, `jms/InvokeService_Response`.

You can set a destination name for both services and references.

Response destination create

Specifies whether to create a response destination for the binding.

Valid values are `ifnotexist` (the default), `always`, and `never`. When the value is `ifnotexist`, the product dynamically creates response destination resources necessary for the SCA composite, if those resources do not exist and relate to the default messaging provider. If the value is `always` and the response destination exists when the SCA composite is added to a business-level application, deployment stops with an error.

Response connection factory JNDI name

Specifies the connection factory that the binding uses to process response messages. The value must be a JNDI name.

You can set a response connection factory for both services and references.

The attributes of this element follow those defined for the destination element. This element is mutually exclusive with the activationSpec element.

Response connection factory create
Specifies whether to create a response connection factory that the binding can use to process response messages.

Valid values are `ifnotexist` (the default), `always`, and `never`. When the value is `ifnotexist`, the product dynamically creates response connection factory resources necessary for the SCA composite, if those resources do not exist and relate to the default messaging provider. If the value is `always` and the response connection factory exists when the SCA composite is added to a business-level application, deployment stops with an error.

Request header properties

Specifies JMS header properties that apply to requests from a reference. You cannot edit the header properties.

If a JMS header property is specified, the property must not appear in the URI.

JMS type
Specifies a JMS type to use in the JMS header property using `@JMSType`.

JMS correlation ID
Specifies a JMS correlation identification to use in the JMS header property using `@JMSCorrelationID`.

JMS delivery mode
Specifies a JMS delivery mode to use in the JMS header property using `@JMSDeliveryMode`.

JMS time to live
Specifies a JMS time to live to use in the JMS header property using `@JMSTimeToLive`.

JMS priority
Specifies a JMS priority to use in the JMS header property using `@JMSPriority`.

Header property
Specifies a value to use for the specified JMS user property.

Response header properties

Specifies JMS header properties that apply to responses from a service for outbound messages. You cannot edit the header properties.

If a JMS header property is specified, the property must not appear in the URI.

JMS type
Specifies a JMS type to use in the JMS header property using `@JMSType`.

JMS correlation ID
Specifies a JMS correlation identification to use in the JMS header property using `@JMSCorrelationID`.

JMS delivery mode
Specifies a JMS delivery mode to use in the JMS header property using `@JMSDeliveryMode`.

JMS time to live
Specifies a JMS time to live to use in the JMS header property using `@JMSTimeToLive`.

JMS priority
Specifies a JMS priority to use in the JMS header property using `@JMSPriority`.

Header property
Specifies a value to use for the specified JMS user property.

Provide HTTP endpoint URL information settings for SCA composites

Use this page to specify endpoint Universal Resource Locator (URL) prefix information for Service Component Architecture (SCA) composites accessed by web service bindings. The information is used to form complete endpoint addresses.

To view this administrative console page, click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_composition_unit_name* > **Provide HTTP endpoint URL information**.

Default SCA URL prefixes

Shows the predefined default endpoint URL prefixes for SCA composites that are accessed by Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS) for service endpoints.

The field shows both unsecure and secure custom endpoint URL values separated by a space. For example:

```
http://theHost:9081 https://theHost:9044
```

For each endpoint URL prefix, the format is *protocol*://*host_name*:*port_number*. The protocol is either `http` or `https`. In this example, *host_name* is `theHost` and *port_number* is the port used in the endpoint URL.

To use the default endpoint URL prefixes, deselect the **Override default SCA URL prefixes** check box.

Override default SCA URL prefixes

Specifies whether to use a custom endpoint URL prefix when the service has a proxied front end. The endpoint URL prefixes are those of the proxy server. You must specify proxied endpoints when deploying services that use the web service binding in a clustered configuration.

To specify a custom endpoint URL prefix, do the following:

1. Select the **Override default SCA URL prefixes** check box.
2. For **HTTP host name**, specify the host name of the unsecure custom endpoint. For example: `myHost`
3. For **HTTP port**, specify the port of the unsecure custom endpoint. For example: `9081`
4. For **HTTPS host name**, specify the host name of the secure custom endpoint. For example: `myHost`
5. For **HTTPS port**, specify the port of the secure custom endpoint. For example: `9044`
6. Click **OK**.

SCA composite component settings

Use this page to view and edit the attributes associated with a Service Component Architecture (SCA) component.

To view this administrative console page, click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_name* > *SCA_composite_component_name*.

Components are configured instances of implementations. Components provide and consume services. More than one component can use and configure the same implementation, where each component configures the implementation differently. For example each component might configure a reference of the same implementation to consume a different service.

An implementation defines the aspects configurable by a component in the form of a component type. The component type is in effect a description of the contract honored by the implementation.

A reference represents a requirement that the implementation has on a service provided by another component.

Component name
Specifies the component name of the attribute.

Implementation
Specifies the name of the class or configuration file that contains the component implementation.

For `implementation.java`, the Java class is shown. For other implementations, the name of the resource identified by the implementation is shown:

- For `implementation.jee`, the archive name is shown.
- For `implementation.spring`, the application context file is shown.
- For `implementation.osgiapp`, the application symbolic name and version is shown.

Type
Specifies the type of attribute. In this case, the type is `Component`.

SCA component services
Specifies the names of the services.

SCA component references
Specifies the names and targets of component references. You can edit the reference target for customization.

SCA component properties
Specifies the **Property Input Source** and **Property Value** for each property.

Options for **Property Input Source** include the following:

- **XPath** indicates the source attribute of the property.
- **File** indicates the file attribute of the property.
- **Value** indicates the property element value.

SCA component reference settings

Use this page to view and edit the attributes associated with a Service Component Architecture (SCA) component reference.

To view this administrative console page, click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_composition_unit_name* > *SCA_composite_component_name* > *reference_name*.

SCA component references within an implementation represent links to services the implementation uses that must be provided by other components in the SCA system. For a composite, you can wire references of components within the composite (component references) to references of the composite (composite references), indicating that the component references must be resolved by services outside the composite.

References use bindings to describe the access methods used to invoke the services.

Under **Additional Properties**, click **View domain** to view a list of services available in the current cell or domain. This can be helpful when updating the **Target** setting value, for example.

Reference name
Specifies the reference name of the attribute.

Type
Specifies the type of attribute. In this case, it is `Reference`.

Reference target URI

Specifies one or more target service uniform resource identifiers (URIs), depending on the multiplicity setting. Each target wires the reference to a component service that resolves the reference. Targets can contain a list of targets separated by a space, in the form `target1 target2`.

Bindings

Specifies the URI of the binding.

Supported bindings include the SCA default binding, enterprise bean (EJB) binding, web service binding, Java Message Service (JMS) binding, Atom binding, and HTTP binding.

SCA component service settings

Use this page to view and edit the attributes associated with a component service.

To view this administrative console page, click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_composition_unit_name* > *SCA_composite_component_name* > *service_name*.

Services are used to publish services provided by implementations, so that they are addressable by other components.

A service published by a component can be provided by a service of a component defined within the component, or it can be provided by a component reference. The latter case allows the republication of a service with a new address or new bindings.

Service name

Specifies the service name of the attribute.

Type

Specifies the type of attribute. In this case, `Service`.

Work manager JNDI name

Specifies the Java Naming and Directory Interface (JNDI) name of the work manager.

Bindings

Specifies the uniform resource identifier (URI) of the binding.

Supported bindings include the SCA default binding, enterprise bean (EJB) binding, web service binding, Java Message Service (JMS) binding, Atom binding, and HTTP binding.

Service provider policy sets and bindings collection for SCA composites

Use this page to attach and detach policy sets to a composition unit, a service provider, its endpoints, or operations of a Service Component Architecture (SCA) composite. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

To view this administrative console page, your composition unit must use web services and support SCA. Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_composition_unit_name* > **Service provider policy sets and bindings** .

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

To attach or detach a policy set or binding, do the following:

1. Select a composition unit, service, endpoint, or operation. The **Composition unit/Service/Endpoint/ Operation** list is nested, indicating parent-child relationships.

2. Click the desired button.

Table 75. Button descriptions. Use the buttons to attach or detach policy sets and to assign policy set bindings.

Button	Resulting action
Attach	Attaches a policy set to the selected composition unit, service, endpoint, or operation. To attach a policy set, select a unit, service, endpoint, or operation and click **Attach** > **_policy_set_option_**. To close the menu list, click **Attach**.
Detach Policy Set	Detaches a policy set from the selected composition unit, service, endpoint, or operation. After the policy set is detached, if there is no policy set attached to an upper-level service resource, the **Attached Policy Set** column displays **None** and the **Binding** column displays **Not applicable**. If there is a policy set attached to an upper-level service resource, the **Attached Policy Set** column displays **_policy_set_name_ (inherited)** and the binding used for the upper-level attachment is applied. The binding name is displayed followed by **(inherited)**.

Table 75. Button descriptions (continued).   Use the buttons to attach or detach policy sets and to assign policy set bindings.

Button	Resulting action
Assign Binding	Assigns a policy set binding to the selected composition unit, service, endpoint, or operation. The options include the following: **Default** Specifies the default binding for the selected service reference, endpoint, or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application-specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used. For any policy set attachment, the run time checks to see if the attachment includes a binding. If so, it uses that binding. If not, the run time checks in the following order and uses the first available default binding: 1. Default general bindings for the server 2. Default general bindings for the domain in which the server resides 3. Default general bindings for the global security domain **New Application Specific Binding** Select this option to create a new application-specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached. **Provider sample** Select this option to use the Provider sample binding. **Provider sample V2** Select this option to use the Provider sample V2 binding when you are using either the Kerberos V5 WSSecurity default or the TrustServiceKerberosDefault policy sets. **Saml Bearer Provider sample** Select this option to use the Saml Bearer Provider sample. The Saml Bearer Provider sample extends the Provider sample binding to support SAML Bearer token usage scenarios. You can use this sample with any of the SAML bearer token default policy sets. **Saml HoK Symmetric Provider sample** Select this option to use the Saml HoK Symmetric Provider sample. The Saml HoK Symmetric Provider sample extends the Provider sample binding to support SAML holder-of-key (HoK) symmetric key token usage scenarios. You can use this sample with one of the SAML HoK Symmetric key default policy sets: either SAML11 HoK Symmetric WSSecurity default or SAML20 HoK Symmetric WSSecurity default.

Composition unit/Service/Endpoint/Operation
Specifies the name of the composition unit and the associated service providers, endpoints or operations.

The Composition unit/Service/Endpoint/Operation column lists the composition unit and the service providers, endpoints, or operations that the composition unit contains.

Attached Policy Set
Specifies the policy set that is attached to a composition unit, service provider, endpoint, or operation.

The Attached Policy Set column can contain the following values:
- **None**. No policy set is attached, either directly or to a higher-level service resource.
- *Policy_set_name*. The name of the policy set that is attached directly to the service resource, for example, WS-I RSP.

- *Policy_set_name* **(inherited)**. The name of the policy set that is not attached directly to a service resource, but that is attached to a higher-level service resource.

When the value in the column is a link, click the link to view or change settings about the attached policy set.

Binding

Specifies the binding configuration that is available for a service provider, endpoint, or operation.

The Binding column can contain the following values:

- **Not applicable**. No policy set is attached, either directly or to a higher-level service resource.
- *Binding_name* or **Default**. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. **Default** is displayed if a policy set is attached directly but the service resource uses the default bindings.
- *Binding_name* **(inherited)** or **Default (inherited)**. A service resource inherits the bindings from an attachment to a higher-level resource.

When the value in the Binding column is a link, click the link to view or change settings about the binding.

References policy sets and bindings collection for SCA composites

Use this page to attach and detach policy sets to a composition unit, a service reference, its endpoints, or operations of a Service Component Architecture (SCA) composite. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service reference can share its current policy configuration.

To view this administrative console page, your composition unit must use web services and support SCA. Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_composition_unit_name* > **References policy sets and bindings**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

To attach or detach a policy set or binding, do the following:

1. Select a composition unit, service, endpoint, or operation. The **Composition unit/Service/Endpoint/ Operation** list is nested, indicating parent-child relationships.
2. Click the desired button.

Table 76. Button descriptions. Use the buttons to attach or detach client policy sets and to assign policy set bindings.

Button	Resulting action
Attach Client Policy Set	Attaches a client policy set to the selected composition unit, service, endpoint, or operation. To attach a policy set, select a composition unit, service, endpoint, or operation and click **Attach Client Policy Set** > *policy_set_option*.
	To close the menu list, click **Attach Client Policy Set**.

Table 76. Button descriptions (continued). Use the buttons to attach or detach client policy sets and to assign policy set bindings.

Button	Resulting action
Detach Client Policy Set	Detaches a client policy set from the selected composition unit, service, endpoint, or operation.
	After the policy set is detached, if there is no policy set attached to an upper-level service resource, the **Attached Client Policy Set** column displays **None** and the **Binding** column displays **Not applicable**.
	If there is a policy set attached to an upper-level service resource, the **Attached Client Policy Set** column displays *policy_set_name* **(inherited)** and the binding used for the upper-level attachment is applied. The binding name is displayed followed by **(inherited)**.
Assign Binding	Assigns a policy set binding to the selected composition unit, service, endpoint, or operation. The options include the following:
	Default Specifies the default binding for the selected service, endpoint, or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application-specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used.
	For any policy set attachment, the run time checks to see if the attachment includes a binding. If so, it uses that binding. If not, the run time checks in the following order and uses the first available default binding:
	1. Default general bindings for the server
	2. Default general bindings for the domain in which the server resides
	3. Default general bindings for the global security domain
	New Application Specific Binding Select this option to create a new application-specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached.
	Client sample Select this option to use the Client sample binding.
	Client sample V2 Select this option to use the Client sample V2 binding when you are using either the Kerberos V5 WSSecurity default or the TrustServiceKerberosDefault policy sets.
	Saml Bearer Client sample Select this option to use the Saml Bearer Client sample. The Saml Bearer Client sample extends the Client sample binding to support SAML Bearer token usage scenarios. You can use this sample with any of the SAML bearer token default policy sets.
	Saml HoK Symmetric Client sample Select this option to use the Saml HoK Symmetric Client sample. The Saml HoK Symmetric Client sample extends the Client sample binding to support SAML holder-of-key (HoK) symmetric key token usage scenarios. You can use this sample with one of the SAML HoK Symmetric key default policy sets: either SAML11 HoK Symmetric WSSecurity default or SAML20 HoK Symmetric WSSecurity default.

Composition unit/Service/Endpoint/Operation

Specifies the name of the composition unit and the associated service references, endpoints or operations.

The Composition unit/Service/Endpoint/Operation column lists the service composition unit and the service references, endpoints, or operations that the composition unit contains.

Attached Client Policy Set

Specifies the policy set that is attached to a composition unit, service reference, endpoint, or operation.

The Attached Client Policy Set column can contain the following values:
- **None**. No policy set is attached, either directly or to a higher-level service resource.
- *Policy_set_name*. The name of the policy set that is attached directly to the service resource, for example, WS-I RSP.
- *Policy_set_name* **(inherited)**. The name of the policy set that is not attached directly to a service resource, but that is attached to a higher-level service resource.

When the value in the column is a link, click the link to view or change settings about the attached policy set.

Binding

Specifies the binding configuration that is available for a service reference, endpoint, or operation.

The Binding column can contain the following values:
- **Not applicable**. No policy set is attached, either directly or to a higher-level service resource.
- *Binding_name* or **Default**. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. **Default** is displayed if a policy set is attached directly but the service resource uses the default bindings.
- *Binding_name* **(inherited)** or **Default (inherited)**. A service resource inherits the bindings from an attachment to a higher-level resource.

When the value in the Binding column is a link, click the link to view or change settings about the binding.

SCA service provider settings

Use this page to manage policy sets for a Service Component Architecture (SCA) web service provider. You can attach and detach policy sets to a service provider, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

To view this administrative console page, your composition unit must use web services and support SCA. Click **Services** > **Service providers** > *service_provider_name*.

Service provider

Specifies the full QName of the service provider. The QName must be in a format that supports the Java class `javax.xml.namespace.QName`.

For the SCA sample business-level application HelloWorldAsync, the service provider name resembles the following:

`{http://websphere.ibm.com/HelloWorldServiceComponent/HelloWorldService}HelloWorldService`

Policy Set Attachments

Specifies the attached policy sets and assigned bindings for services, endpoints, or operations in the service provider.

To attach or detach a policy set or to assign bindings with system-specific configurations, do the following:

1. Select a service, endpoint, or operation. The **Service/Endpoint/Operation** list is nested, indicating parent-child relationships.
2. Click the desired button.

Table 77. Button descriptions.   Use the buttons to attach or detach policy sets and to assign policy set bindings.

Button	Resulting action
Attach	Attaches a policy set to the selected service, endpoint, or operation. To attach a policy set, select a service, endpoint, or operation and click **Attach** > *policy_set_option*. To close the menu list, click **Attach**.
Detach Policy Set	Detaches a policy set from the selected service, endpoint, or operation. After the policy set is detached, if there is no policy set attached to an upper-level service resource, the **Attached Policy Set** column displays **None** and the **Binding** column displays **Not applicable**. If there is a policy set attached to an upper-level service resource, the **Attached Policy Set** column displays *policy_set_name* **(inherited)** and the binding used for the upper-level attachment is applied. The binding name is displayed followed by **(inherited)**.

Table 77. Button descriptions (continued).   Use the buttons to attach or detach policy sets and to assign policy set bindings.

Button	Resulting action
Assign Binding	Assigns a policy set binding to the selected service, endpoint, or operation. The options include the following: **Default** Specifies the default binding for the selected service reference, endpoint, or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application-specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used. For any policy set attachment, the run time checks to see if the attachment includes a binding. If so, it uses that binding. If not, the run time checks in the following order and uses the first available default binding: 1. Default general bindings for the server 2. Default general bindings for the domain in which the server resides 3. Default general bindings for the global security domain **New Application Specific Binding** Select this option to create a new application-specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached. **Provider sample** Select this option to use the Provider sample binding. **Provider sample V2** Select this option to use the Provider sample V2 binding when you are using either the Kerberos V5 WSSecurity default or the TrustServiceKerberosDefault policy sets. **Saml Bearer Provider sample** Select this option to use the Saml Bearer Provider sample. The Saml Bearer Provider sample extends the Provider sample binding to support SAML Bearer token usage scenarios. You can use this sample with any of the SAML bearer token default policy sets. **Saml HoK Symmetric Provider sample** Select this option to use the Saml HoK Symmetric Provider sample. The Saml HoK Symmetric Provider sample extends the Provider sample binding to support SAML holder-of-key (HoK) symmetric key token usage scenarios. You can use this sample with one of the SAML HoK Symmetric key default policy sets: either SAML11 HoK Symmetric WSSecurity default or SAML20 HoK Symmetric WSSecurity default.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Service/Endpoint/Operation

Specifies the name of the service and the associated service providers, endpoints or operations.

The Service/Endpoint/Operation column lists the service and the service providers, endpoints, or operations that the service contains.

Attached Policy Set

Specifies the policy set that is attached to a service provider, endpoint, or operation.

The Attached Policy Set column can contain the following values:

- **None**. No policy set is attached, either directly or to a higher-level service resource.

- **Policy_set_name**. The name of the policy set that is attached directly to the service resource, for example, WS-I RSP.
- **Policy_set_name (inherited)**. The name of the policy set that is not attached directly to a service resource, but that is attached to a higher-level service resource.

When the value in the column is a link, click the link to view or change settings about the attached policy set.

Binding

Specifies the binding configuration that is available for a service provider, endpoint, or operation.

The Binding column can contain the following values:

- **Not applicable**. No policy set is attached, either directly or to a higher-level service resource.
- **Binding_name** or **Default**. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. **Default** is displayed if a policy set is attached directly but the service resource uses the default bindings.
- **Binding_name (inherited)** or **Default (inherited)**. A service resource inherits the bindings from an attachment to a higher-level resource.

When the value in the Binding column is a link, click the link to view or change settings about the binding.

About policy set bindings

In this release, there are two types of bindings: application-specific bindings and general bindings. Composition units can use both application-specific bindings and general bindings.

Application-specific bindings

You can create application-specific bindings only at a policy set attachment point. These bindings are specific to, and constrained by, the characteristics of the defined policy. Application-specific bindings can provide configuration for advanced policy requirements such as multiple signatures; however, these bindings are reusable only within an application. Also, application-specific bindings have very limited reuse across policy sets.

When you create an application-specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding, and fully configure the bindings for each policy that you add. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application-specific bindings.

For service providers, you can create application-specific bindings only by selecting **Assign Binding** > **New Application Specific Binding**, on the Service providers policy sets and bindings collection page, for service provider resources that have an attached policy set. Similarly, for service clients, you can create application-specific bindings only by selecting **Assign Binding** > **New Application Specific Binding**, on the Service clients policy sets and bindings collection page, for service client resources that have an attached policy set.

General bindings

You can configure general bindings to be used across a range of policy sets and they can be reused across applications and for trust service attachments. Although general bindings are highly reusable, they cannot provide configuration for advanced policy requirements such as multiple signatures. There are two types of general bindings: general provider policy set bindings and general client policy set bindings.

You can create general provider policy set bindings by clicking **Services** > **Policy sets** > **General provider policy set bindings** > **New** in the general provider policy sets panel, or by clicking **Services** > **Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. For details about defining and managing service client or provider bindings, see the related links. General provider policy set bindings might also be used for trust service attachments.

SCA service client settings

Use this page to manage policy sets for a Service Component Architecture (SCA) web service client. You can attach and detach policy sets to a service reference, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service reference can share its current policy configuration.

To view this administrative console page, your composition unit must use web services and support SCA. Click **Services** > **Service clients** > *service_client_name*.

Service client

Specifies the full QName of the service client. The QName must be in a format that supports the Java class `javax.xml.namespace.QName`.

For the SCA sample business-level application HelloWorldAsync, the service client name resembles the following:

`{http://websphere.ibm.com/HelloWorldServiceComponent/HelloWorldService}HelloWorldCallbackService`

This SCA application has the product web service namespace, `http://websphere.ibm.com/`, and the service name in its service client name.

Policy Set Attachments

Specifies the attached policy sets and assigned bindings for services, endpoints, or operations in the service client.

To attach or detach a policy set or to assign bindings with system-specific configurations, do the following:

1. Select a service, endpoint, or operation. The **Service/Endpoint/Operation** list is nested, indicating parent-child relationships.
2. Click the desired button.

Table 78. Button descriptions.   Use the buttons to attach or detach client policy sets and to assign policy set bindings.

Button	Resulting action
Attach Client Policy Set	Attaches a client policy set to the selected service, endpoint, or operation. To attach a policy set, select a service, endpoint, or operation and click **Attach Client Policy Set** > *policy_set_option*. To close the menu list, click **Attach Client Policy Set**.
Detach Client Policy Set	Detaches a client policy set from the selected service, endpoint, or operation. After the policy set is detached, if there is no policy set attached to an upper-level service resource, the **Attached Client Policy Set** column displays **None** and the **Binding** column displays **Not applicable**. If there is a policy set attached to an upper-level service resource, the **Attached Client Policy Set** column displays *policy_set_name* **(inherited)** and the binding used for the upper-level attachment is applied. The binding name is displayed followed by **(inherited)**.

Table 78. Button descriptions (continued). Use the buttons to attach or detach client policy sets and to assign policy set bindings.

Button	Resulting action
Assign Binding	Assigns a policy set binding to the selected service, endpoint, or operation. The options include the following: **Default** Specifies the default binding for the selected service, endpoint, or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application-specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used. For any policy set attachment, the run time checks to see if the attachment includes a binding. If so, it uses that binding. If not, the run time checks in the following order and uses the first available default binding: 1. Default general bindings for the server 2. Default general bindings for the domain in which the server resides 3. Default general bindings for the global security domain **New Application Specific Binding** Select this option to create a new application-specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached. **Client sample** Select this option to use the Client sample binding. **Client sample V2** Select this option to use the Client sample V2 binding when you are using either the Kerberos V5 WSSecurity default or the TrustServiceKerberosDefault policy sets. **Saml Bearer Client sample** Select this option to use the Saml Bearer Client sample. The Saml Bearer Client sample extends the Client sample binding to support SAML Bearer token usage scenarios. You can use this sample with any of the SAML bearer token default policy sets. **Saml HoK Symmetric Client sample** Select this option to use the Saml HoK Symmetric Client sample. The Saml HoK Symmetric Client sample extends the Client sample binding to support SAML holder-of-key (HoK) symmetric key token usage scenarios. You can use this sample with one of the SAML HoK Symmetric key default policy sets: either SAML11 HoK Symmetric WSSecurity default or SAML20 HoK Symmetric WSSecurity default.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Service/Endpoint/Operation

Specifies the name of the service and the associated service references, endpoints or operations.

The Service/Endpoint/Operation column lists the service and the service references, endpoints, or operations that the service contains.

Attached Client Policy Set

Specifies the policy set that is attached to a service reference, endpoint, or operation.

The Attached Client Policy Set column can contain the following values:

- **None**. No policy set is attached, either directly or to a higher-level service resource.
- *Policy_set_name*. The name of the policy set that is attached directly to the service resource, for example, WS-I RSP.
- *Policy_set_name* **(inherited)**. The name of the policy set that is not attached directly to a service resource, but that is attached to a higher-level service resource.

When the value in the column is a link, click the link to view or change settings about the attached policy set.

Binding

Specifies the binding configuration that is available for a service reference, endpoint, or operation.

The Binding column can contain the following values:

- **Not applicable**. No policy set is attached, either directly or to a higher-level service resource.
- *Binding_name* or **Default**. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, MyBindings1. **Default** is displayed if a policy set is attached directly but the service resource uses the default bindings.
- *Binding_name* **(inherited)** or **Default (inherited)**. A service resource inherits the bindings from an attachment to a higher-level resource.

When the value in the Binding column is a link, click the link to view or change settings about the binding.

About policy set bindings

In this release, there are two types of bindings: application-specific bindings and general bindings. Composition units can use both application-specific bindings and general bindings.

Application-specific bindings

You can create application-specific bindings only at a policy set attachment point. These bindings are specific to, and constrained by, the characteristics of the defined policy. Application-specific bindings can provide configuration for advanced policy requirements such as multiple signatures; however, these bindings are reusable only within an application. Also, application-specific bindings have very limited reuse across policy sets.

When you create an application-specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding, and fully configure the bindings for each policy that you add. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application-specific bindings.

For service providers, you can create application-specific bindings only by selecting **Assign Binding** > **New Application Specific Binding**, on the Service providers policy sets and bindings collection page, for service provider resources that have an attached policy set. Similarly, for service clients, you can create application-specific bindings only by selecting **Assign Binding** > **New Application Specific Binding**, on the Service clients policy sets and bindings collection page, for service client resources that have an attached policy set.

General bindings

You can configure general bindings to be used across a range of policy sets and they can be reused across applications and for trust service attachments. Although general bindings are highly reusable, they

cannot provide configuration for advanced policy requirements such as multiple signatures. There are two types of general bindings: general provider policy set bindings and general client policy set bindings.

You can create general provider policy set bindings by clicking **Services** > **Policy sets** > **General provider policy set bindings** > **New** in the general provider policy sets panel, or by clicking **Services** > **Policy sets** > **General client policy set bindings** > **New** in the general client policy set and bindings panel. For details about defining and managing service client or provider bindings, see the related links. General provider policy set bindings might also be used for trust service attachments.

Example: Creating an SCA business-level application with the console

You can add many different types of artifacts to business-level applications. For example, you can add applications or modules, Java archives (JAR files), data in compressed files, and other business-level applications. This example describes how to create an empty business-level application and then add a Service Component Architecture (SCA) JAR file to the application using the administrative console.

Before you begin

In a product installation, verify that the target server is configured. As part of configuring the server, determine whether your application files can run on your deployment target. You must deploy SCA composite assets of a business-level application to a Version 8.0 or later server (target) or to a Version 7.0 target that is enabled for the Feature Pack for SCA.

Download the `helloworld-ws-asynch.jar` SCA sample file from a product download site:

1. Go to the **Samples, Version 8.5** information center.
2. On the **Downloads** tab, click **FTP** or **HTTP** in the **Service Component Architecture** section.
3. In the authentication window, click **OK**.
4. In the **SCA.zip** compressed file, go to the `SCA/installableApps` directory and download the `helloworld-ws-asynch.jar` file.

About this task

For this example, use the administrative console to create a business-level application named HelloWorldAsync that has an SCA JAR file, `helloworld-ws-asynch.jar`, as an asset.

Procedure

1. Create an empty business-level application named `HelloWorldAsync`.
    a. Click **Applications** > **New Application** > **New Business Level Application**.
    b. On the New application page, specify the name `HelloWorldAsync`, optionally add a description, and then click **Apply**.
    c. On the page that is displayed, click the **Save** link.

    The name is shown in the list of applications on the Business-level applications page. Because the application is empty, its status is `Unknown`.

2. Import the SCA JAR asset.
    a. Click **Applications** > **New Application** > **New Asset** in the console navigation tree.
    b. On the Upload asset page, specify the asset package to import, `helloworld-ws-asynch.jar`, and click **Next**.
       The JAR file is in the *app_server_root*/`installableApps` directory.
    c. On the Select options for importing an asset page, click **Next** to accept the default values.
    d. On the Summary page, click **Finish**.
    e. On the Adding asset to repository page, if messages show that the operation completed, click **Manage assets**.

f. On the Assets page, click the **Save** link.

   The file name displays in the list of assets.
3. Add the SCA JAR asset as a composition unit of the business-level application.
   a. Click **Applications** > **Application Types** > **Business-level applications**.
   b. On the Business-level applications page, click the `HelloWorldAsync` application name.
   c. On the business-level application settings page, click **Add** > **Add Asset**.
   d. On the Add page, select the `helloworld-ws-asynch.jar` asset composition unit from the list of available units, and then click **Continue**.
   e. On the Set options page, click **Next** to accept the default values.
   f. On the Map composition unit to a target page, specify a target server that supports SCA composites, and then click **Next**.
   g. On the Define relationship with existing composition units page, click **Next** to accept the default values.
   h. On the Map virtual host page, click **Next** to accept the default values.
   i. On the Summary page, click **Finish**.

   Several messages are displayed. A message having the format `Completed res=[WebSphere:cuname=helloworldws]` indicates that the addition is successful.

   During deployment of the composition unit, you can view the Uniform Resource Identifier (URI) for composite level service of some bindings, along with the service name and binding type. Only the URI is editable. The product does not validate the URI.
   j. If the addition is successful, click **Manage application**.
   k. On the business-level application settings page, click **Save**.

   The asset name and type displays in the list of deployed assets. If you click on the asset name, the composition unit settings page displays, with the asset name in the **SCA Composite Components** list.
4. Start the HelloWorldAsync business-level application.
   a. Click **Applications** > **Application Types** > **Business-level applications**.
   b. On the Business-level applications page, select the check box beside HelloWorldAsync.
   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

Optionally examine, and possibly use in applications, other SCA sample files in the `SCA/installableApps` directory of the downloadable SCA samples.

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. If SCA composite assets do not start, ensure that each asset is mapped to a deployment target that supports SCA composites.

If an asset composition unit uses an Enterprise JavaBeans (EJB) binding and does not start because it has a `non-WebSphere target of "null"`, delete the asset composition unit and add it again to the business-level application. Specify a target that supports SCA composites when you add the asset to the business-level application. You cannot change the target after deployment.

If the SCA application uses security, the target server or cluster must be in the global security domain.

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.

Procedure

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.

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.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

If the application contains Service Component Architecture (SCA) composites and does not start, check for the following problems:

- If SCA composite assets do not start, ensure that each asset is mapped to a Version 8 deployment target or to a Version 7 deployment target that supports SCA composites.
- If an asset composition unit uses an Enterprise JavaBeans (EJB) binding and does not start because it has a `non-WebSphere target of "null"`, delete the asset composition unit and add it again to the business-level application. Specify a target that supports SCA composites when you add the asset to the business-level application. You cannot change the target after deployment.
- If the `META-INF/sca-deployables` directory has multiple SCA composite files and the application does not start because the product `cannot obtain the CompUnitInfoLoader value`, place only the file that contains the composite in the `META-INF/sca-deployables` directory. You can place the other composite files anywhere else within the archive.

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.

Procedure

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 the changes that you want to make to your application. Also, determine whether the changed application can run on your deployment targets.

The administrative console Server pages show the versions for deployment targets.

If you want to change a composition unit that contains an enterprise bundle archive (EBA) asset, see Modifying the configuration of an EBA asset.

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.

Procedure

- 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 page, 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.

       Before updating a shared library, you must manually stop all Java EE applications that depend on that shared library. After updating the shared library, manually restart the Java EE applications. Java EE applications do not automatically restart when a shared library is updated.
    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 page, select a unit from the list of available units, and then click **Continue**.
    6. On the Set options page, change the composition unit settings as needed, and then click **Next**.
    7. On the Map composition unit to a target page, change the deployment target as needed, and then click **Next**.

       This page 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.

Updating SCA composite artifacts

You can view and update Service Component Architecture (SCA) composite artifacts in business-level applications.

Before you begin

Add an SCA artifact as a composition unit to a business-level application.

About this task

You can view and update the following SCA composite artifacts:
- Composite level property definition
- Composite level component property definition
- Composite level component reference definition

You can view and update SCA composite artifacts using the administrative console or the wsadmin tool. This topic describes how to view and update SCA composite artifacts using the administrative console.

Procedure

1. Go to the composition unit settings page for an SCA composite artifact in a business-level application.

   Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *SCA_deployed_asset_composition_unit_name*.

   The composition unit settings page for an SCA composite artifact has fields that are not shown on the composition unit settings page for a non-SCA artifact:
   - SCA composite components
   - SCA composite properties
   - SCA composite wires

2. Click on a name link in one of these SCA fields to view the settings for an SCA artifact.

   The SCA fields display None instead of a name link if the composition unit does not have that particular type of SCA composite.

3. Optional: Update a SCA composite setting value.

   a. Change an existing setting value for the SCA artifact.

   b. Click **OK**.

   The setting value is updated.

Viewing and updating SCA composites in HelloWorldAsync

"Example: Creating an SCA business-level application with the console" on page 535 describes how to create the HelloWorldAsync business-level application. This application contains an SCA artifact, helloworldws, as a composition unit. You can view and update settings for SCA composites in the helloworldws composition unit using the console.

1. Go to the composition unit settings page for the helloworldws composition unit in the HelloWorldAsync business-level application.

Click **Applications** > **Application Types** > **Business-level applications** > **HelloWorldAsync** > **helloworldws**.

From the composition unit settings page, you can view information associated with helloworldws, as well as update composite settings.

2. Click on a link for the SCA artifact to be viewed or updated.

For example, click on the HelloWorldServiceComponent link under **SCA composite components** and, in the page that displays, click on the HelloWorldService link under **Service**. In the Component service settings page that displays, you can specify a setting value for the service.

3. If you update a setting value for the SCA artifact, click **OK**.

What to do next

Save the changes to your administrative configuration.

On multiple-server products, when saving the configuration, synchronize the configuration with the nodes where the application is expected to run.

Viewing SCA composite definitions

You can view information on the definition of a Service Component Architecture (SCA) composite in the administrative console.

Before you begin

The SCA composite must be a composition unit in a business-level application.

About this task

The composite definition provides data on the composite, such as component names and service references. The View composite page displays the composite definition of an SCA deployed asset composition unit.

Procedure

1. Go to the View composite page.

Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *SCA_deployed_asset_name* > **View composite**.

2. Optional: Click **Expand All** or **Collapse All** to more easily browse the page.

Results

The View composite page displays the contents of the composition unit definition.

Example

Suppose the HelloWorldAsync business-level application provided as a sample with the product is installed. Click **Applications** > **Application Types** > **Business-level applications** > **HelloWorldAsync** > **helloworldws** > **View composite**.

The View composite page displays configuration information resembling the following:

```
<composite targetNamespace="http://helloworld" name="helloworldws" >
  <component name="AsynchTranslatorComponent" >
    <implementation.java class="helloworld.impl.AsynchTranslatorComponent" />
      <service name="AsynchTranslatorService">
        <interface.java interface="helloworld.AsynchTranslatorService"
            callbackInterface="helloworld.HelloWorldCallback" />
```

```
            <binding.ws/>
              <callback>
                  <binding.ws/>
              </callback>
        </service>
    </component>
</composite>
```

What to do next

Browse the page to ensure that it contains the intended configuration information.

Viewing SCA domain information

You can view information on Service Component Architecture (SCA) composites in an SCA domain in the administrative console.

Before you begin

The SCA composite must be a composition unit in a business-level application.

About this task

Viewing SCA domain information enables you to see on one console page information on all components in an SCA domain. The View domain page displays information on available services in the current domain.

Procedure

1. Go to the View domain page.

   Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *SCA_deployed_asset_name* > **View domain**.

2. Optional: Click **Expand All** or **Collapse All** to more easily browse the page.

Results

The View domain page lists information on components in the current domain.

Example

Suppose the HelloWorldAsync business-level application provided as a sample with the product is installed. Click **Applications** > **Application Types** > **Business-level applications** > **HelloWorldAsync** > **helloworldws** > **View domain**.

The View domain page displays information resembling the following:

```
<domain name="myCell02">
  <component name = "AsynchTranslatorComponent"
        mapTarget = "WebSphere:cell=myCell02,node=myNode02,server=server1">
    <service name = "AsynchTranslatorService">
      <interface.java interface = "helloworld.AsynchTranslatorService"/>
    </service>
    <reference name = "AsynchTranslatorService" target = ""/>
    <httpurlendpoints name = "endpoints" uri = ""/>
  </component>
</domain>
```

What to do next

Browse the page to ensure that it contains the intended information.

You can export the same domain information to a file using the exportCompositeToDomain command. See "Exporting SCA domain information using scripting."

Viewing and editing JMS bindings on references and services of SCA composites

You can view information on a Java Message Service (JMS) binding for a Service Component Architecture (SCA) composite in the administrative console. The JMS bindings page of the console displays the settings of a `binding.jms` element. You can use the console page to edit resource and response resource settings.

Before you begin

Configure an SCA composite that uses JMS bindings and add that composite as a composition unit to a business-level application. To view and edit settings for a JMS binding on references, the composite must define a JMS binding in the reference. Similarly, to view and edit settings for a JMS binding on services, the composite must define a JMS binding in the service.

About this task

The JMS bindings page enables you to see JMS binding settings for an SCA composite in the administrative console. The information shown is similar to that shown by running the `viewCompUnit` wsadmin scripting command to view the SCA composite composition unit. However, the console page shows JMS binding settings only, and does not show other information on the composition unit that running `viewCompUnit` returns.

After deployment of an SCA composite that uses JMS bindings, you can edit JMS binding resource or response resource settings that specify Java Naming and Directory Interface (JNDI) names on the JMS bindings page. The editing capabilities are similar to those of the `editCompUnit` scripting command. To edit settings, the JMS resource must exist. The product does not dynamically create JMS resources when you edit a composition unit.

Procedure

1. Go to the JMS bindings page.

   To view this page, your composition unit must support a JMS binding reference or service.

   a. Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *deployed_asset_composition_unit_name*.

   b. From the composition unit settings page for SCA composites, select to view references or services:

      • For an SCA component reference, click **SCA Composite Components** > *SCA_component_reference_name* > **SCA Component References** > *reference_name* > **Bindings** > **JMS binding**.

      • For an SCA component service, click **SCA Composite Components** > *SCA_component_service_name* > **SCA Component Services** > *service_name* > **Bindings** > **JMS binding**.

2. View the JMS binding settings for the composition unit.

3. Edit the **Destination JNDI name**, **Activation specification JNDI name**, **Connection factory JNDI name**, **Response destination JNDI name**, or **Response connection factory JNDI name** settings as needed and click **Apply**.

Results

The JMS bindings page displays property settings for a JMS binding. If you change resource or response resource JNDI name values, the changed values are shown.

Example

Suppose that you have a business-level application named `MyJmsBLA`, which has an SCA composite named `MySCAComposite`. This composite uses a JMS binding on service named `JmsService`, in a composition unit named `myJmsBindingCU`.

1. Click **Applications** > **Application Types** > **Business-level applications** > **MyJmsBLA** > **myJmsBindingCU** > **SCA Composite Components** > **MySCAComposite** > **SCA Component Services** > **JmsService** > **Bindings** > **JMS binding**.

   The JMS bindings page displays general binding settings and settings for a JMS binding on services.

2. Edit the value specified for **Activation specification JNDI name**, and click **Apply**.

   The JMS bindings page displays the changed value.

What to do next

Browse the page to ensure that it contains the binding information.

For information on JMS binding settings, refer to the online help for the JMS bindings page or Section 1.4 of the SCA JMS Binding specification, Version 1.00.

Exporting WSDL and XSD documents

You can export Web Services Description Language (WSDL) and XML schema definition (XSD) documents used by a Service Component Architecture (SCA) composition unit to a location of your choice.

Before you begin

Your SCA business-level application must contain one or more composition units that use a WSDL or XSD document.

A WSDL document is a file that provides a set of definitions that describe a web service in WSDL, an Extensible Markup Language (XML)-based description language.

An XSD document is an instance of an XML schema written in the XML schema definition language. The document has the extension `.xsd`. The prefix `xsd` in the XML elements of an XSD document indicates the XML schema namespace.

About this task

You can export WSDL and XSD documents that are used by an SCA composition unit using the administrative console. In previous releases, you had to use the `exportWSDLArtifacts` command to export WSDL and XSD documents.

On the composition unit settings page for an SCA composite, click the **Export WSDL and XSD documents** link and then specify the target location for the files.

The product extracts from the selected composition unit the WSDL and XSD files that are required for web services client development. The files are for the services exposed by the web service binding, `binding.ws`.

Procedure

1. Go to the composition unit settings page for the SCA composite.

   Click **Applications** > **Application Types** > **Business-level applications** > *application_name* > *SCA_deployed_asset_name*.

2. Click the **Export WSDL and XSD documents** link.

3. From the displayed dialog, specify the target directory to which to save the documents.

Results

The WSDL and XSD documents are copied to the target directory.

Example

Suppose you want to export WSDL or XSD documents in the HelloWorldAsync business-level application that the product provides as a sample. Complete the following actions in the administrative console:

1. Click **Applications** > **Application Types** > **Business-level applications** > **HelloWorldAsync** > **helloworldws** > **Export WSDL and XSD documents**.
2. Using the displayed dialog, specify a directory that exists on your computer.

The product adds the `helloworldws_WSDLArtifacts.zip` file to the specified directory. The `helloworldws_WSDLArtifacts.zip` file has one WSDL file, `AsynchTranslatorComponentTranslatorService_wsdlgen.wsdl`.

What to do next

Examine the exported files to ensure that they contain the intended WSDL and XSD documents.

You can export WSDL and XSD documents using the exportWSDLArtifacts command. See Exporting WSDL and XSD documents using scripting.

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.

Procedure

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, delete each composition unit of the application. Deployed assets and business-level applications can be composition units of a business-level application.

      Select one or more composition units and click **Delete**.

      For Service Component Architecture (SCA) business-level applications that use an OSGi implementation in an enterprise bundle archive (EBA) composition unit, delete the SCA composition unit before attempting to delete the EBA composition unit, otherwise the product returns an error. An EBA composition unit that is used for the implementation cannot be deleted until the SCA composition unit that uses the `implementation.osgiapp` is deleted.
   c. On the Delete composition unit from Business-level application page, confirm the deletion and click **OK**.

d. Repeat steps b and c until the business-level application that you want to delete has no more composition units.

   Deleting a composition 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 page, 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.

Deleting the HelloWorldAsync business-level application

"Example: Creating an SCA business-level application with the console" on page 535 describes how to create the HelloWorldAsync business-level application. You can delete this application using the console.

1. Go to the Business-level applications page and, if HelloWorldAsync is running, change its status to Stopped.

   a. Click **Applications** > **Application Types** > **Business-level applications**.

   b. Select HelloWorldAsync.

   c. Click **Stop**.

2. Go to the business-level applications settings page for HelloWorldAsync and delete the helloworldws composition unit.

   a. Click **Applications** > **Application Types** > **Business-level applications** > **HelloWorldAsync**.

   b. From **Deployed assets**, select helloworldws.

   c. Click **Delete**.

   d. On the Delete composition unit from Business-level application page, confirm the deletion and click **OK**.

   e. Click the **Save** link to save the changes.

3. From the business-level applications page, delete the HelloWorldAsync application.

   a. Click **Applications** > **Application Types** > **Business-level applications**.

   b. Select HelloWorldAsync.

   c. Click **Delete**.

   d. On the Delete business-level application page, click **OK**.

   e. Click the **Save** link to save the changes.

4. Optionally, from the Assets page, delete the helloworld-ws-asynch.jar asset from the asset repository.

   a. Click **Applications** > **Application Types** > **Assets**.

   b. Select helloworld-ws-asynch.jar.

   c. Click **Delete**.

   d. On the Delete asset page, click **OK**.

   e. Click the **Save** link to save the changes.

What to do next

If using the administrative console **Delete** options does not fully delete a business-level application or its composition units, you can delete the business-level application and its composition 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 13. 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.

Procedure

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 corrctly, 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.

Procedure

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.

Procedure

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("c:\sources\test5.zip"); //Change to the directory of your sources.
```

```
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 STORAGETYE_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 = "c:\websphere\asset\installDir\asset1.zip";
```

```
        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.

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.

Procedure

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.

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.

Procedure

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.

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.

Procedure

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 = "c:\myInstalledAssets\asset1.zip";
```

```
        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 perameter.
            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 paramter 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.

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.

Procedure

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.

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.

Procedure
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.
```

Windows

```
            DownloadFile filename = new DownloadFile("c:\assets\asset1.zip");
```

Linux   **HP-UX**   **Solaris**   **AIX**

```
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.

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.

Procedure

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.

Procedure

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
// manage:.
//
// 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.

Procedure

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.

Table 79. Business-level application status descriptions. Read the descriptions to learn about application status.

Status	Description
ExecutionState.STARTED	All composition unit instances within the scope of the query have been started.
ExecutionState.STOPPED	All composition unit instances within the scope of the query have not been started or have been stopped.
ExecutionState.PARTIAL_START	Some composition unit instances within the scope of the query have a status of ExecutionState.STARTED and some have a status of ExecutionState.STOPPED.
ExecutionState.UNKNOWN	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.

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.

Procedure

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 blaID 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.

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.

Procedure

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 blaID 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 compostion 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 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 paramaeter value, but no cuID parameter value. To list control operations for a composition unit, specify both a blaID parameter value and a cuID 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.

Procedure

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.

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 blaID parameter to specify the business-level application that you are viewing.

Perform the following tasks to view a business-level application using programming.

Procedure

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.

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 cuID 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.

Procedure

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 blaID and cuID 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.

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.

Procedure

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 blaID 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.

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. Wether 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.

Procedure

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 = "c:\assets\abc.txt";
```

```
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.

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.

Procedure

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.

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.

Procedure

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.

If you are editing a composition unit to update a shared library, you must first manually stop all Java EE applications that depend on that shared library. After updating the shared library, manually restart the Java EE applications. Java EE applications do not automatically restart when a shared library is updated.

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 asynchronoous 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 compostion unit is updated.
// Valid values are DEFAULT, ALL, NONE
String stepName = "CUOptions";
CommandStep step = ((TaskCommand) cmd).gotoStep(stepName);

// Composition Unit description:
String description = "cu1 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.

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.

Procedure

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 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 cuID 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.

Procedure

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 cuID parameters are required parameters. The cuID 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.

Chapter 14. Troubleshooting deployment

When you are having problems deploying an application, perform some basic diagnostics and verify your system configuration to solve the problem.

Before you begin

Try to install your application on a product server. Ensure that your application can be installed to the deployment target. For example, if your application contains modules that support Java Platform, Enterprise Edition (Java EE) 6 or use a Version 8 product feature or API, you must install the application to a Version 8 deployment target.

About this task

Determine which of the following steps apply to the deployment problem and read the suggested topics.

Procedure
- If you cannot install the application, troubleshoot problems deploying applications.

  See the topics on application deployment problems and troubleshooting tips.
- If you can install the application but it does not start, troubleshoot problems starting applications.

  See the topics on application deployment and startup problems.
- If your application contains many classes with annotations and takes a long time to deploy, reduce annotation searches to speed up deployment.

  See the topic on reducing annotation searches during application deployment.
- If you cannot uninstall the application, see the topic on application uninstallation problems.

What to do next

If the topics in this information center do not resolve the deployment problem, examine current information available from IBM Support on known problems and their resolution. IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see topic on 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?
- "Application does not display " on page 638
- "Unable to save a deployed application" on page 638
- ▐ Windows ▐ "java.lang.RuntimeException: Failed_saving_bytes_to_wor_ERROR_ error in the assembly tool, administrative console or the wsadmin tool" on page 639
- "WASX7015E error running wsadmin command $AdminApp installInteractive or $AdminApp install" on page 639
- "Cannot install a CMP or BMP entity bean in an EJB 3.0 module" on page 639
- "Data definition language (DDL) generated by an assembly tool throws SQL error on target platform" on page 640
- "ADMA0004E: Validation failed" on page 640
- "Cannot load resource WEB-INF/ibm-web-bnd.xmi in archive file" on page 640

- "Timeout!!!error displays when attempting to install an enterprise application in the administrative console " on page 641
- "NameNotFoundException message when deploying an application that contains an EJB module" on page 641
- **Windows** "During application installation, the call to EJB deploy causes an exception" on page 642
- "Compilation errors and EJB deploy fails when installing an EJB JAR file generated for Version 5.x or earlier" on page 642
- "After installing the application onto a different machine, the application does not run" on page 642
- "A single file replaces all application files during application update" on page 642

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.

Application does not display

Application installed 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.

Linux On the SuSE9 or other Linux platform, you can either increase the number of files that can be opened to resolve the problem or you can modify the application to close files with disciplines. To increase the number of files that you can open at the same time, run the following command in the shell before invoking the process that needs to open a number of files:

`ulimit -n `*number_of_files*

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.

Windows

java.lang.RuntimeException: Failed_saving_bytes_to_wor_ERROR_ error in the assembly tool, administrative console or the wsadmin tool

If you see this error when attempting to generate deployed code in an assembly tool, installing an application or module in the administrative console, or using the wsadmin tool to install an application or module, the file path length of the temporary system file might be exceeded.

To verify this problem, check the `TEMP` and `TMP` environment variables for your system. Long environment variables add path length to the file names accessed by the EJB deployment tool.

To resolve the problem:
1. Stop all WebSphere Application Server processes and close all DOS prompts.
2. Set the `TEMP` and `TMP` environment variables to something short, for example `C:\TMP` and `C:\TEMP`.
3. Reinstall the application.

Otherwise, try rebooting and redeploying or reinstalling the application.

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:

AIX   HP-UX   Linux   Solaris   Windows

```
WASX7015E: Exception running command:
"$AdminApp installInteractive C:/Documents and Settings/
  myUserName/Desktop/MyApp/myapp.ear";
exception information:
com.ibm.bsf.BSFException: error while
evaluating 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.

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:
* **AIX** **HP-UX** **Solaris** **Windows** Edit the DDL in the vi editor, removing the Ctl-M character at the beginning of each line.
* **Linux** Regenerate the deployment code for the application EAR file on a Linux platform.

ADMA0004E: Validation failed

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 beannameBean 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.

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

"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.

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.

During application installation, the call to EJB deploy causes an exception

When you specify that the EJB deployment tool be run during application installation and if installation fails with the error command line too long, the problem is that the deployment command generated during installation exceeds the character limit for a command line on the Windows platform.

To work around this problem, you can reduce the length of the EAR file name, reduce the length of the JAR file name within the EAR file, reduce the class path or other options specified for deployment, or change the %TEMP% location of the Windows system to make its path shorter.

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 or later product, it is compiled using the Version 6.0 or later 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:
- "Application startup problems" on page 648.
- "Web resource is not displayed" on page 653.
- "A client program does not work" on page 652.

You can use the following administrative console pages to inspect the configuration of your applications and JMS resources:
- For a view of the JMS resources for a given application, see the following page: ae/AppToSIBRefs_DetailForm.dita.
- For a view of the applications and JMS resources for a given default messaging provider destination, see the following page: ae/AppsFromSIBRefs_DetailForm.dita.

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 webpage.

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.

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" on page 644
- "File serving problems" on page 644
- "Graphics do not appear in the JSP file or servlet output" on page 645
- "SRVE0026E: [Servlet Error]-[Unable to compile class for JSP file" on page 645
- "After modifying and saving a JSP file, the change does not show up in the browser " on page 647
- "Error message: /jspname.jsp(9,0) Include: Mandatory attribute page missing" on page 647
- "The Java source generated from a JSP file is not retained in the temp directory " on page 647
- "Error Enterprise Application [application name you typed in] not found" on page 647
- "Translation problem with non-English browser input" on page 648
- "Scroll bars do not appear around items in the browser window" on page 648
- "Error Page cannot be displayed, server not found or DNS error" on page 648

The following note applies to the `ibm-web-ext.xmi` references throughout this topic:

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:
- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.

- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

  However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

  The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

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.
   - Retry the web request.

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 *install_root*/installedApps/*nodename*/*appname*.ear directory or *install_root*/installedApps/*nodename*/*appname*Network.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 *install_root*`/config/cells/`*nodename* or *nodename*`Network/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. The product 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 *install_root*`/installedApps/`*nodename*`/`*appname*`.ear` directory or *install_root*`/installedApps/`*nodename*`/`*appname*`Network.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: `<img SRC="banner.gif">`, not `<img SRC="banner.gif">`.

- 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 *install_root*`/config/cells/`*nodename* or *nodename*`Network/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;
       ^
```

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

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 `c:\WebSphere\ApplicationServer\ 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

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.

Error message: /jspname.jsp(9,0) Include: Mandatory attribute page missing

The error "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

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.

Error "Enterprise Application [application name you typed in] not found"

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 *install_root*`\config\cells\`*node_name*`Network\applications`.

For example:
- `"JspBatchCompiler -enterpriseapp.name sampleApp.ear/deployments/sampleApp"` is correct, as opposed to
- `"JspBatchCompiler -enterpriseapp.name  sampleApp"`, which is incorrect.

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"

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?
- "WSVR0100W: An error occurred initializing, application_name java.lang.NullPointerException when starting a migrated application" on page 649
- A "java.lang.ClassNotFoundException: classname Bean_AdderServiceHome_04f0e027Bean" on page 649 error occurs
- A "ConnectionFac E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter" on page 650 error occurs
- "NMSV0605E: A Reference object looked up from the context... error when starting an application" on page 650.
- A parsing error when running an application that uses the JSF configuration occurs.
- "A Page Not Found, Array Index Out of Bounds, or other error when an updated application restarts" on page 651

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**.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see IBM Support troubleshooting information.

WSVR0100W: An error occurred initializing, *application_name* java.lang.NullPointerException when starting a migrated application

After you migrate an enterprise application to Version 8.0, the application might not start. Attempts to start the application result in an error such as `WSVR0100W: An error occurred initializing,` *application_name* `java.lang.NullPointerException`.

Examine the `deployment.xml` file of the migrated application, and remove `targetMapping` statements such as the following:

```
<targetMappings xmi:id="DeploymentTargetMapping_1279594183813" enable="true"/>"
```

Then, try starting the application again. The Version 8.0 runtime has an application validation process that might not support migrated `targetMappings` settings.

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. If you do not have access to the target server, you can specify a false location such as `/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 stand-alone 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 the topic on 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.

ConnectionFac 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 UrlContextHel 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 Providers** > **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.

Parsing error when running an application that uses the JSF configuration

If you are using double-byte characters in the profile name, you receive a parsing error when running an application that uses the JavaServer Faces (JSF) configuration. The problem is related to the JSF configuration that is part of the `jsf-ibm.jar`, which is included when building JSF applications in Rational Application Developer. The configuration files are referencing entities from inside the main `faces-config.xml` file.

Avoid using double-byte characters when you create a profile.

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.

Reducing annotation searches during application deployment

Enterprise applications that contain many classes with annotations might take a long time to deploy. Java EE 5 introduced annotations to add metadata to Java classes. Because of performance issues associated with reflection and because classes are not always loadable at deployment, bytecode scanning technology is used to retrieve annotation metadata. Java EE 5 or later applications with many classes might experience long deployment times because every class within the application is inspected during deployment. You can reduce the number of annotations to inspect by specifying the modules and Java packages to ignore for annotations processing in the `amm.filter.properties` file or by configuring system properties.

Before you begin

Install an application that supports Java Platform, Enterprise Edition (Java EE) 5 or later on a product server. If deployment is unreasonably slow and you will be deploying this application again in the future, complete a procedure in this topic to reduce the number of classes that are searched for annotations during deployment.

About this task

The product provides a configurable filtering function to reduce the number of classes that are searched for annotations. You can identify which modules or Java packages to ignore for annotations processing through two properties:
- Ignore-Scanning-Archives
- Ignore-Scanning-Packages

A default set of values is provided in the `amm.filter.properties` file in *app_server_root*/`properties`. The property values provide both coarse and fine grained control over the search scope for annotations

processing. Use of the Ignore-Scanning-Archives property reduces deployment time more than use of the Ignore-Scanning-Packages property. The syntax for the Ignore-Scanning-Archives and Ignore-Scanning-Packages properties follows the comma-separated value convention. No wildcard or regular expressions are permitted and values are case-sensitive.

The default set of values can be changed by an administrator or augmented by a user using one of the following steps.

Procedure

- Place an `amm.filter.properties` file in the *profile_root*/`properties` directory.
- Use system properties to supply values for the Ignore-Scanning-Archives and Ignore-Scanning-Packages properties.
  - The `com.ibm.ws.amm.scan.context.filter.archives` system property supplies values for the Ignore-Scanning-Archives property.
  - The `com.ibm.ws.amm.scan.context.filter.packages` system property supplies values for the Ignore-Scanning-Packages property.

  See the topic on Java virtual machine custom properties.
- Add Ignore-Scanning-Archives and Ignore-Scanning-Packages entries to the application manifest, `META-INF/MANIFEST.MF`.

  Note: When updating the application manifest, follow line-length limitations and other constraints for the manifest.
- Add Ignore-Scanning-Archives and Ignore-Scanning-Packages entries to the module manifest.

  Note: When updating the module manifest, follow line-length limitations and other constraints for the manifest.

What to do next

Install the application again. If deployment continues to be slow, specify more modules and Java packages to ignore.

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 stand-alone 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 stand-alone 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 stand-alone 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.

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.

Windows Look in the Services panel for the service corresponding to your HTTP server, and verify that the state is **Started**. If not, start it. If the service does not start, try starting it manually from the command prompt. If you are using IBM HTTP Server, the command is `IHS_install_dir\apache` .

AIX **HP-UX** **Linux** **Solaris** Run the **ps -ef | grep httpd** command. There should be several processes running with a name of "httpd". If not, start your HTTP server manually. If you are using IBM HTTP Server, the command is `IHS_install_dir/bin/apachectl start`.

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 website 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 administrative console, click **Servers** > **Server Types** > **WebSphere application servers** > *application_server* > **Web container**.
   b. Under the Additional Properties of the web container, click **HTTP Transports**. You see the ports listed for virtual hosts served by the application server.
   c. 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 **Servers** > **Server Types** > **WebSphere application servers** > *application_server* > **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 15. Troubleshooting administration

Use this information if you are having problems with administrative functions.

Procedure

- 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

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

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 658
- "When starting or stopping a server using a wsadmin interactive scripting session, you receive an exception indicating read timed out" on page 659
- "Problems starting or using the administrative console or wsadmin utility" on page 659
- "The AdminConfig.parents method cannot be used to obtain the valid parent types of some configuration types" on page 660

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 once 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 prior to 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 prior to 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. 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 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, refer to the following 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 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 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 correct server.
- 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"

The AdminConfig.parents method cannot be used to obtain the valid parent types of some configuration types

When using the AdminConfig.parents method to obtain the allowed parent types for a specific configuration type, for example, "JavaVirtualMachine", an informational message may be returned instead of the desired value.

```
wsadmin>print AdminConfig.parents( "JavaVirtualMachine")
```

The following informational message is returned:

```
WASX7351I: The parents command cannot be used to find the parents of type "JavaVirtualMachine"
```

.

The following function demonstrates one possible work around for the AdminConfig.parents() method that deals with the issue previously mentioned:

```
#---------------------------------------------------------------------
# Name: parentTypes()
# Role: To search for configuration types that may contain objects of
#       the specified type should the result of calling
```

```
#       AdminConfig.parents( Type )
#       be:
#       WASX7351I: The parents command cannot be used to find the parents of type ...
# Note: Unlike the AdminConfig.list() command, this routine will not
#       raise an InvalidConfigDataTypeException for an unknow datatype
#----------------------------------------------------------------------
def parentTypes( Type ) :
    import os
    WSAStypes = AdminConfig.types().splitlines()
    if Type in  WSAStypes :
        result = AdminConfig.parents( Type )
        if result.startswith( 'WASX7351I' ) :
            result = []
            for thisType in WSAStypes :
                if AdminConfig.attributes( thisType ).find( Type ) > -1 :
                    result.append( thisType )
            result = (os.linesep).join( result )
    else :
        print 'parentTypes error: unknown / unrecognized type:', Type
        result = None
    return result
```

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 has been identified and documented using the links in the Diagnosing and fixing problems: Resources for learning topic. 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:

- 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. Prior to 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.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

What kind of problem are you having?
- "Internal Server Error, Page cannot be found, 404, or similar error occurs trying to view the administrative console" on page 662

- "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. Before opening a problem report, see the Support page:

- http://www.ibm.com/software/webservers/appserv/was/support/

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 take one of the following actions:
  - Run the **startServer server1** command for the Windows platform from a command prompt in the `install_dir`\bin directory, or the **./startServer.sh server1** command for operating systems such as AIX or Linux.
  - Click the **Start the server** link from the First steps console.
  - Start WebSphere Application Server as a service or from the Start menu, if you are using a Windows operating system.
- 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 troubleshooting installation.

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

An error can occur when a single user that uses multiple instances of the Mozilla to log into the administrative console

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.

Mozilla user selects a check box on a collection table 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.

Mozilla user enters an invalid ID or password 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: **HP-UX** 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 thehttp_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:
      - **Windows** On a Windows platform, search for the file `gsk7ssl.dll`

- **AIX** **HP-UX** **Solaris** On a UNIX platform, search for a libgsk7*.so files in the `/usr/lib` directory.

  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 the http_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, seeWeb 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 problems with the wsadmin scripting tool

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 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
- 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
- ▶ AIX ▶ HP-UX ▶ Solaris WASX7022E: Problem running command "import sys" -- exception information: com.ibm.bsf.BSFException: unable to load language

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' Xchange. For specific details relating to the Java implementation of Tcl, refer to The Tcl/Java Project.
- 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.

"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 changed in 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`

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.

▶ AIX   ▶ HP-UX   ▶ Solaris
WASX7022E: Problem running command "import sys" -- exception information: com.ibm.bsf.BSFException: unable to load language

This problem may be caused by a limitation on some UNIX platforms, for example, Linux, when attempting to use the Jython language.

To workaround this problem, perform the following steps:
1. Check the number of open files that you are allowed to have on the machine, for example:

   `ulimit -a`
2. Check the number of open files that you have set on your machine. The default value is `1024`.
3. Change it to a higher number, for example:

   `ulimit -n 2048`
4. Try to use the wsadmin tool again with the Jython language.

Application Server start or restart 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

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

- Browse the Application Server log files for clues. The log files are located by default in:
  - `Linux` `AIX` `HP-UX` `Solaris` *profile_root*/logs/*server_name*/SystemErr.log and SystemOut.log
  - `Windows` *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.
- `Linux` `AIX` `HP-UX` `Solaris` The `tail -f` *profile_root*/ logs/*server_name*/SystemOut.log command is a convenient way to watch the progress of the server.
- 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, in a base or non-distributed configuration on Windows systems, 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 non-root user ID to run application servers, verify that:
  - The non-root user has write access to the *app_server_root*/temp directory.
  - The JVM has write access to *app_server_root*/config/plugin-cfg.xml file.
  - The non-root user has access to the `logs` directory.
- 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*.

An error message might appear in the SystemOut.log after 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 the command:

   ```
   netstat -a
   ```
2. Restart the server

Message "DiscoveryService.sendQuery" exception appears in the FFDC log file

When you start a deployment manager, the deployment manager attempts to discover any configured node agents within its cell. If the deployment agent does not discover the node agents in the cell, it writes an exception to the first failure data capture (FFDC) log file for each node agents that the deployment manager does not discover. If the node agents are not suppose to be running, you can ignore the exception. If the node agents are suppose to be running, the FFDC log file might contain additional information that will help you determine why the deployment manager cannot discover the node agents even though the node agents are suppose to be running.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems. Before opening a problem report, see the Support page:
- http://www.ibm.com/software/webservers/appserv/was/support/

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 topic provides troubleshooting support for a variety of problems relating to using command-line tools.

What kind of problem are you having?
- b
- 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.

- `[Windows]` With Windows service, there is no indication when a server is already started.
- The stopServer command fails to stop the server because the system cannot create a connector to an invalid hostname.
- `[Windows]` `[AIX]` Entering passwords on the command line

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`

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log`, `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

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`

`[Windows]`
No indication that a server is already started on Windows

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.

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 (for example, on AIX, HP-UX, Linux, or Solaris computers, 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")
     ```

Windows **AIX**
Entering passwords on the command line

There are operating system differences when specifying environment variables and for directory notation. When using the Windows command line, replace $variable with %variable% for environment variables and replace each forward slash (/) with a backslash (\) in directory paths. If you are using the bash shell on a Windows system, you can use the UNIX conventions.

On Unix type operating systems, it is usually better to always encapsulate string values associated with all tags in double quotes (""). You will also need to escape, using a '\' character, the following characters if they appear in the string values: '!', '\', '"', '`'. This will prevent some command line shells from interpreting specific characters within these values.

Some examples of usage follow:
```
gsk7capicmd -keydb -create -db "/tmp/key.kdb" -pw "j\!jj"
```

Note: When prompted by gsk7capicmd for a value (for example a password) quoting the string and adding the escape characters should not be done because the shell is no longer influencing this input.

Windows
```
stopserver server1 -username fn_wasadmin -password $mypass35%% -trace
```

AIX
```
stopserver server1 -username fn_wasadmin -password '$mypass35%' -trace
```

Note: Use single quotation marks to specify that what is inside the single quotation marks is to be used. Using single quotation marks can be helpful for any characters that might otherwise have special meaning (like the dollar sign ("$") or the backslash ("\")). Single quotation marks treat these characters literally. In this case, the password being supplied is $mypass35% .

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.

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.

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

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 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).

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 and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

administrative console   177
  accessing help   187
  buttons   165
  command assistance   188
    actions   190
  features   168
  logging off   179
  product installation   178
  product uninstallation   178
  saving changes   181
  settings   182
    preferences   185
    scopes   186
  specifying preferences   182
  starting   179
  usage   165
Ant   253
Ant plug-in   134
application defined data sources   158
application files
  adding   395
  changing   395
Application Monitor
  dynamic updates   141
Application Server
  runtime environments   128
applications
  administering
    targets   271
  AppManagement MBean   463
  bindings   348
  common deployment framework   272
  configuration
    binary location usage   359
    enterprise applications   346
  deployment
    targets   271
  deployment descriptors   376
  deployment targets   385
  disabling automatic start   381
  dynamic reloading   393
  editing applications   442
  enterprise applications   354
  export copy   406
  export data   407
  exporting from servers   405
  hot deployment   393
  installation bindings   288
  management   437
  managing   413, 414
  partial applications
    adding to   445
    deleting   445
    updating   445
  removing application files   408
  resolving configuration conflicts   403

applications *(continued)*
  session sharing   439
  settings   356
    binary   360
    installation   287
    startup behavior   358
    updates   388
  starting   380
  startup configuration   357
  status   382
  stopping   380
  system applications   272
  uninstallation   409
    administrative console   408
  updating
    administrative console   387
  updating files   384
assets   478
  deleting   483
  exporting   483
  importing   472
  managing   478
  settings   474
    adding   491
    updating   481
    uploading   473
  updating   479
authentication
  JASPI enablement
    applications   316

B

backend ID
  settings   336
business-level applications   468, 488
  administering   549
  assets   470
  composition units   470
  creating   484, 500
    administrative console   485
  deleting   545
  programming   551
    deleting   628
    editing   619
    listing   589
    starting   579
    stopping   582
  SCA   503
    administrative console   505, 535
  settings   489, 495
  starting   537
  stopping   538
  updating   538