WebSphere® Application Server for i5/OS, Version 6.1

Migrating, coexisting, and interoperating

Note

Before using this information, be sure to read the general information under “Notices” on page 121.

Compilation date: April 20, 2006

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 e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-0206.

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

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

Chapter 1. Overview and new features for migrating, coexisting, and interoperating

Use the links provided in this topic to learn about the migration, coexistence, and interoperability features.

"Overview of migration, coexistence, and interoperability" on page 72

This topic describes common topologies that you can install with the product.

Deprecated and removed features

Various features have been either deprecated or removed in WebSphere Application Server product offerings since Version 5.0.

If a feature is listed here as deprecated, IBM intends to remove this capability in a future release of the product. Generally, IBM will not remove a feature until at least two major releases or three full years (whichever is longer) after the release in which it is deprecated. For example, features that were deprecated in WebSphere Application Server Version 5.0, Version 5.0.1, or Version 5.0.2 will not be removed from the product until after Version 6.0 because both Version 5.0.x and Version 5.1.x are considered to be major releases. In rare cases, it might become necessary to remove features sooner; such cases are indicated clearly and explicitly in the descriptions of these deprecated features in this article.

Deprecated and removed features

See the following topics to find more information on deprecated and removed features:

- "Deprecation list"
- "Removal list" on page 11

Deprecation list

The following tables summarize deprecated features by version and release. The tables indicate what is being deprecated, such as application programming interfaces (APIs), scripting interfaces, tools, wizards, publicly exposed configuration data, naming identifiers, and constants. Where possible, the tables also indicate the recommended migration action.

This article contains the following deprecation tables:

- "Features deprecated in Version 6.1"
- "Features deprecated in Version 6.0.2" on page 4
- "Features deprecated in Version 6.0" on page 4
- "Features deprecated in Version 5.1.1" on page 7
- "Features deprecated in Version 5.1" on page 7
- "Features deprecated in Version 5.0.2" on page 10
- "Features deprecated in Version 5.0.1" on page 11
- "Features deprecated in Version 5.0" on page 11

Features deprecated in Version 6.1

Application Programming Model Features	
Deprecation	**Recommended Migration Action**

The setDatabaseDefaultIsolationLevel(int) method in the com.ibm.websphere.rsadapter.DataStoreHelperMetaData class is deprecated.	Start using the following instead: `public final void setDatabaseDefaultIsolationLevel ( int helperDefaultLevel, int cusDefinedWasDefaultIsoLevel )`
The following class and interface in the Mediation Framework runtime are deprecated: • com.ibm.websphere.sib.mediation.handler.SIMessageContextException class • com.ibm.websphere.sib.mediation.messagecontext.SIMediationBean MessageContext interface	Replace all uses of the com.ibm.websphere.sib,.mediation.handler.SIMessageContextException class with the com.ibm.websphere.sib.mediation.handler.MessageContextException class. Replace all uses of the com.ibm.websphere.sib.mediation.messagecontext.SIMediationBean MessageContext interface with an equivalent interface. WebSphere Application Server does not provide an implementation of this interface.
The following WebContainer MBean functions are deprecated: • startTransports • stopTransports • restartWebApplication	Begin moving to the channel framework. The channel framework provides the TransportChannelService MBean, which is more flexible and has more methods than the current WebContainer transport-related methods.
Support for deploying container-managed entity beans to a generic SQL database is deprecated.	If an application uses SQL92 or SQL99 because the application has to run with different relational databases, use the IBM tooling to generate deployed code for each database vendor or version that the application might use. At installation time, specify the database vendor or version that will used with WebSphere Application Server.
The IBM WebSphere Studio tools runtime support provided by the following classes (which were used to leverage VAJava tooling) is deprecated: • com.ibm.webtools.runtime.AbstractStudioServlet • com.ibm.webtools.runtime.BuildNumber • com.ibm.webtools.runtime.NoDataException • com.ibm.webtools.runtime.StudioPervasiveServlet • com.ibm.webtools.runtime.TransactionFailureException • com.ibm.webtools.runtime.WSUtilities	Rearchitect your applications to use standard J2EE coding conventions.
The CUSTOM_HELPER constant field in the com.ibm.websphere.rsadapter.DataStoreHelper class API is deprecated.	If you create your own DataStoreHelper implementation class, do not invoke setHelperType(DataStoreHelper.CUSTOM_HELPER). Instead, let the HelperType value be set by the implementation class from which it inherits.
J2EE Resource Features	
Deprecation	**Recommended Migration Action**
Support for the ability to connect from either an application server or a J2EE application client to the JMS Server component of the embedded messaging feature in WebSphere Application Server Version 5 is deprecated. This deprecation includes the following: • Ability to define JMS resource definitions for the Version 5 default messaging provider • Ability to establish connections from client applications that are either running in a Version 5 environment or utilizing Version 5 default messaging provider resource definitions	Perform the following actions: 1. Ensure that any JMS Server messaging providers that are hosted on WebSphere Application Server Version 5.1 application servers are moved onto Version 6.0 or later application servers. This task is handled automatically when you migrate a Version 5.x server to Version 6.0 or later. 2. Change all JMS resource definitions to use the new Version 6 default messaging provider instead of the Version 5 default messaging provider.
System Administration Features	
Deprecation	**Recommended Migration Action**
WebSphere administrative (wsadmin) scripting support for the Jacl language is deprecated.	Begin converting your existing Jacl syntax wsadmin scripts into Jython syntax wsadmin scripts, and use Jython syntax for any new wsadmin scripting. For more information, go to **Application Server Toolkit > Administering applications > Developing automation scripts** in this information center.
The **clientUpgrade** command is deprecated.	No migration action is necessary.
The **wasprofile** command is deprecated.	Use the **manageprofiles** command.
The following **WASPostUpgrade** command parameters are deprecated: • -import *xmi_data_file* • -substitute *"key1=value1*[;*key2=value2*;[...]]*"*	No migration action is necessary.

The Cloudscape datastore helper (com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper) and Cloudscape Network Server datastore helper (com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper) as well as their types in DataStoreHelper are deprecated.	For existing configurations, no migration action is necessary. The migration utility changes the deprecated Cloudscape helpers to Derby helpers. For new configurations, use the Derby datastore helpers and types instead of the Cloudscape datastore helpers.
The Derby Network Server Using Universal JDBC Driver Provider, which uses the db2jcc.jar file, is deprecated.	Start using the Derby Network Server Using Derby Client provider, which uses the derbyclient.jar file and has the benefit of supporting XA.
The DB2 Legacy CLI-based Type 2 JDBC Driver provider is deprecated.	Start using the DB2 Universal JDBC Driver Provider.
Logical pool distribution support (com.ibm.websphere.csi.ThreadPoolStrategy.LogicalPoolDistribution) is deprecated.	No migration action is necessary. When this function is removed, however, all custom ORB properties that you specified for it will be ignored. The custom ORB properties of interest are com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.*.
ORB thread pool configuration as part of the Server object in the server.xml file is deprecated.	Use the thread pool configuration that is part of the ServerIndex object in the serverindex.xml file.
The JVM system property com.ibm.websphere.sendredirect.compatibility is deprecated.	Begin modifying your applications to redirect non-relative URLs, those starting with a forward slash ("/"), relative to the servlet container root (*web_server_root*) instead of the web application context root. See the *Java Servlet 2.4 Specification*, available for download at http://jcp.org/aboutJava/communityprocess/final/jsr154/, for information on how sendRedirect should behave.
The Web container PageList Servlet custom extension is deprecated, including the following classes: • com.ibm.servlet.ClientList • com.ibm.servlet.ClientListElement • com.ibm.servlet.MLNotFoundException • com.ibm.servlet.PageListServlet • com.ibm.servlet.PageNotFoundException	Rearchitect your applications to use javax.servlet.filter classes rather than com.ibm.servlet classes. Starting with the Java Servlet 2.3 specification, javax.servlet.filter classes give you the capability to intercept requests and examine responses. They also allow provide chaining functionality as well as functionality for embellishing or truncating responses.
The following datasource custom properties are deprecated: • dbFailOverEnabled • connRetriesDuringDBFailover • connRetryIntervalDuringDBFailover	Replace the deprecated properties with new properties. • Use validateNewConnection instead of dbFailOverEnabled. • Use validateNewConnectionRetryCount instead of connRetriesDuringDBFailover. • Use validateNewConnectionRetryInterval instead of connRetryIntervalDuringDBFailover. **Note:** If both properties coexist, the new properties take precedence.
Security Features	
Deprecation	**Recommended Migration Action**
The Simple WebSphere Authentication Mechanism (SWAM) is deprecated.	Use the Lightweight Third Party Authentication (LTPA) mechanism.
The LoginHelper CORBA authentication helper function (com.ibm.ws.security.util.LoginHelper) is deprecated.	Migrate to the Java Authentication and Authorization Service (JAAS) programming model. For information on this migration, see "Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS)" on page 57.
Performance Features	
Deprecation	**Recommended Migration Action**
Support for the Java Virtual Machine Profiler Interface (JVMPI) is deprecated along with three related JVM runtime counters • ObjectMovedCount • ObjectFreedCount • ObjectAllocateCount	Begin moving to the Java Virtual Machine Tool Interface (JVMTI). See JVM Tool Interface (JVMTI).
Problem Determination Features	
Deprecation	**Recommended Migration Action**
The message ID format that is used in WebSphere Application Server Version 6.0.x and earlier is deprecated. The message prefixes for log files were not previously registered with the primary message registry. WebSphere Application Server Version 6.1 and later use compliant message prefixes in the output logs.	Use the **convertlog** command and the MessageConverter class to assist you in migrating tools that rely on the older message format.

The com.ibm.etools.logging.util plug-in, the logutil.jar file, is deprecated. • Logging facility Logging facility used for logging Java primitives and complex objects to named loggers; configurable with predefined filtering levels, Logging Agent and file sinks, and output formats through an API, Eclipse plug-in manifest, or Eclipse preference panel • Logging agent XML-based messaging agent used in conjunction with the IBM Agent Controller to write log and trace XML records to a logging service remotely attachable through an API or Test and Performance Tools Platform (TPTP), formally Hyades, Eclipse workbench • Problem determination artifacts and messages Original implementation of the Manageability (M12) Model Problem Determination Architecture Version 1.5 and Problem Determination Artifacts Common Data Model specification used for capturing and encoding log and trace data • Distributed correlator service (DCS) Distributed correlator service that is used for instrumenting correlation identifiers for correlating log and trace data across one or more hosts • Java client bindings Java client bindings used to communicate with the IBM Agent Controller to launch local and remote processes, attach to running processes, and monitor active agents in a secure client environment	Begin moving plug-ins and application code using configuration files, classes, methods, or variables in the com.ibm.etools.logging.util plug-in to the following replacements: • Logging facility **Replacement:** Java Logging APIs in Java Version 1.4.0+; Logging Agent support for the Java Logging APIs provided in TPTP and Common Logging (com.ibm.etools.common.logging/logging.jar) • Logging agent **Replacement:** TPTP Logging Agent (org.eclipse.hyades.logging.core/hlcore.jar) • Problem determination artifacts and messages **Replacement:** Common Base Event Version 1.0.1 specification and TPTP implementation (org.eclipse.hyades.logging.core/hlcbe101.jar) • Distributed correlator service (DCS) **Replacement:** TPTP Correlation Service (org.eclipse.hyades.execution.correlation/hcorrelation.jar) • Java client bindings **Replacement:** TPTP Java Client Bindings (org.eclipse.hyades.execution/hexl.jar) For more information, see the com.ibm.etools.logging.util\doc\IBM_Logging_Utilities_Migration_Guide.html document.

Features deprecated in Version 6.0.2

Application Programming Model Features	
Deprecation	**Recommended Migration Action**

The following methods from class com.ibm.websphere.runtime.ServerName are deprecated:
`initialize(java.lang.String cell, java.lang.String node, java.lang.String server)`
`was390Initialize(byte[] a_stoken, String a_printable_stoken, String a_jsabpref, int a_pid, int an_asid, String a_jsabjbnm)`
`was390Initialize(byte[] a_stoken, java.lang.String a_printable_stoken, java.lang.String a_jsabpref, int a_pid, int an_asid, java.lang.String a_jsabjbnm, java.lang.String a_smcasid)`
	These methods are for WebSphere Application Server runtime use only. Applications should not call these methods.

Support for HTTP transport configuration is deprecated.	Begin moving to channel-based transport.
Performance Features	
Deprecation	**Recommended Migration Action**
The com.ibm.websphere.cache.DistributedLockingMap interface is deprecated.	Do not use the com.ibm.websphere.cache.DistributedLockingMap interface because this interface is not supported by the WebSphere Application Server runtime.
The TYPE_DISTRIBUTED_LOCKING_MAP constant that is defined in the com.ibm.websphere.cache.DistributedObjectCache class is deprecated.	Do not use the TYPE_DISTRIBUTED_LOCKING_MAP constant that is defined in the com.ibm.websphere.cache.DistributedObjectCache class because this constant is not supported by the WebSphere Application Server runtime.

Features deprecated in Version 6.0

Application Programming Model and Container Support Features	
Deprecation	**Recommended Migration Action**

Support for the following tsx tags in the JavaServer Pages (JSP) engine are deprecated:
• repeat
• dbconnect
• dbquery
• getProperty
• userid
• passwd
• dbmodify
	Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL). JSTL is supported in WebSphere Application Server Version 6.0, and the tag library is shipped with the product. Use the following table as a guideline for converting tsx tags to JSTL tags:
	`tsx tag` `JSTL tag`
	`tsx:repeat` `c:forEach`
	`tsx:dbconnect` `sql:setDataSource`
	`tsx:dbquery` `sql:query`
	`tsx:getProperty` `Use standard EL syntax; c:out value="${book.title}" for example, where book is the current index in the result set`
	`tsx:userid` `Use the user attribute of the setDataSource tag`
	`tsx:passwd` `Use the password attribute of the setDataSource tag`
	`tsx:dbmodify` `sql:update`

The following back-end IDs are deprecated: • SQL92 (1992 SQL Standard) • SQL99 (1999 SQL Standard)	Use other back-end IDs.
Application Services Features	
Deprecation	**Recommended Migration Action**
The JRAS Extensions API is deprecated. No further enhancements are planned for JRAS support.	Use the equivalent function in the java.util.logging package (JSR47).
The UDDI Version 2 EJB interface to the UDDI Registry is deprecated.	There is no replacement for the EJB interface. This interface is included in WebSphere Application Server Version 6.0 for compatibility with Version 5.x. Users do not need to take any specific actions and can continue to use the Version 2 EJB API, but they should be aware that it does not include any UDDI functionality that is new to UDDI Version 3 and the interface might be removed in a future release of WebSphere Application Server.
The UDDI4J Version 2 class library, the uddi4jv2.jar file, is deprecated.	Start using the Version 3 UDDI APIs. A client library is provided to simplify constructing and sending UDDI Version 3 requests from Java. This is the IBM UDDI Version 3 Client for Java, provided in uddiv3client.jar. The UDDI4J APIs can still be used, but you should be aware that they do not provide access to any of the new UDDI Version 3 functionality and they might be removed in a future release of WebSphere Application Server.
All of the low-level UDDI Utility Tools (UUT) APIs, such as BusinessStub, ServiceStub, and so on, are deprecated. These are all replaced with the high-level PromoterAPI interface in the com.ibm.uddi.promoter package.	Start using the PromoterAPI interface in the com.ibm.uddi.promoter package in place of these low-level APIs, which will be removed in a future release of WebSphere Application Server. The PromoterAPI provides the same functionality at a higher level of abstraction.
The following methods in the J2EE Connector Architecture runtime are deprecated: • com.ibm.ws.management.descriptor.xml.ConnectionFactory.xml (getPoolContents and getAllPoolContents methods) • com.ibm.websphere.j2c.ConnectionManager interface • com.ibm.websphere.j2c.ConnectionEventListener interface Also, container-managed authentication aliases on a J2C Connection Factory or datasource are deprecated.	The methods are replaced as follows: • getPoolContents and getAllPoolContents are replaced with showPoolContents and showAllPoolContents. • ConnectionManager interface is replaced with J2EE Connector Architecture 1.5 LazyAssociatableConnectionManager interface. • ConnectionEventListener interface is replaced with J2EE Connector Architecture 1.5 LazyEnlistableConnectionManager interface. For container-managed authentication aliases, specify the container-managed credentials in the application's resource binding information.
The ApplicationProfile property on the WorkManager panel in the administrative console is deprecated.	See the information center for the differences between application profiling in Version 5.x and Version 6.0.x.
Two items from the DataSource panel in the administrative console are deprecated: • Container-Managed Authentication Alias • DefaultPrincipleMapping	Define Container-Managed Authentication Alias and DefaultPrincipleMapping properties on the Resource Reference.
All classes in the com.ibm.websphere.servlet.filter package are deprecated: • ChainedRequest • ChainedResponse • ChainerServlet • ServletChain	Rearchitect your applications to use javax.servlet.filter classes rather than com.ibm.websphere.servlet.filter classes. Starting from the Servlet 2.3 specification, javax.servlet.filter classes give you the capability to intercept requests and examine responses. They also allow you to achieve chaining functionality, as well as embellishing and truncating responses.

MIME filtering is deprecated. MIME filters were first introduced in WebSphere Application Server Version 3.5 as a way for servlets to embellish, truncate, and modify the responses generated by other servlets, based on the MIME types of the output content.	The javax.servlet.*filters*, which were introduced in the Servlet 2.3 specification, allow users to plug in filters that can intercept requests to and responses from servlets. They also have the capability to modify content flowing in either direction. The javax.servlet.*filters* maintain all the functionality of MIME filters. javax.servlet.*filters* are standard APIs, and are supported by all compliant application servers. Refer to the Servlet 2.3 specification for more information.
Container-managed persistence (CMP) entity beans configured with method level access intent might run into data access problems, like deadlock. Therefore, the method level access intent is deprecated.	Reconfigure CMP entity beans to use bean level access intent, or reconfigure Application profiles with WebSphere Application Server Tool (AST).
All the methods and fields in com.ibm.websphere.product.product and com.ibm.websphere.product.buildInfo classes are deprecated. Therefore, the following methods from com.ibm.websphere.product.WASProduct class (which involves com.ibm.websphere.product.product and com.ibm.websphere.product.buildInfo objects) are deprecated: • public product getProductByFilename(String basename) • public product getProductById(String id) • public boolean productPresent(String id) • public boolean addProduct(product aProduct) • public boolean removeProduct(product aProduct) • public Iterator getProducts() • public Iterator getProductNames() • public String loadVersionInfoAsXMLString(String filename) • public String getProductDirName() • public static String computeProductDirName()	Use the following supported methods from com.ibm.websphere.product.WASDirectory: • public WASProductInfo getWASProductInfo(String id) • public boolean isThisProductInstalled(String id) • public WASProductInfo[] getWASProductInfoInstances() • public String getWasLocation() Also, instead of getting product information (name, version, build level, build date) from the old WASProduct API (com.ibm.websphere.product.WASProduct), you should now use the following methods in the WASDirectory class to get that information: • com.ibm.websphere.product.WASDirectory.getName(String) • com.ibm.websphere.product.WASDirectory.getVersion(String) • com.ibm.websphere.product.WASDirectory.getBuildLevel(String) • com.ibm.websphere.product.WASDirectory.getBuildDate(String)
Data access beans, which are shipped with WebSphere Application Server in the databeans.jar file, are deprecated.	Instead of using data access beans, you should use Service Data Objects (SDO).
The `reloadInterval` and `reloadingEnabled` attributes of the IBM deployment descriptor extensions are deprecated, including both the WAR file extension (WEB-INF/ibm-web-ext.xmi) and the application extension (META-INF/ibm-application-ext.xmi).	Instead of using deployment descriptor extensions, you should use the reload enable and interval options provided during application deployment.
The com.ibm.websphere.servlet.session.UserTransactionWrapper API is deprecated.	There is no replacement for this API. The UserTransaction object can be placed directly into the HTTP session without using a wrapper.
Security Features	
Deprecation	**Recommended Migration Action**
SOAP-Security (XML digital signature) based on Apache SOAP implementation is deprecated.	Instead of using SOAP-Security, you should migrate your application to JSR-109 implementation of Web service. Also, migrate (reconfigure your application) to use WSS (Web Service Security) 1.0 implementation.
Web Service Security (WSS) draft 13 specification-level support is deprecated in favor of the WSS 1.0 implementation.	Applications should be migrated to the supported WSS 1.0 standard. The draft-level support does not provide interoperability with some third party vendors, as the message level has been changed between the draft and the WSS 1.0 implementation. WSS 1.0 is only supported in J2EE 1.4 applications. Therefore, you need to migrate applications to J2EE 1.4 first. The next step is to use AST/RAD tooling to reconfigure WSS for the migrated application. There is no automatic migration of WSS in this release of AST/RAD tooling for Version 6.0; the migration has to be done manually. The following SPI has also been deprecated: com.ibm.wsspi.wssecurity.config.KeyLocator You need to migrate your implementation to the new SPI for WSS 1.0 support in Version 6.0: com.ibm.wsspi.wssecurity.keyinfo.KeyLocator Finally, the Java Authentication and Authorization Service (JAAS) LoginModule implementation needs to be migrated to the new programming model for JAAS LoginModule in Version 6.0.
The Secure Authentication Service (SAS) IIOP security protocol is deprecated.	Use the Common Secure Interoperability Version 2 (CSIv2) protocols.

The Secure Authentication Service (SAS) CORBA security programming APIs are deprecated	Migrate from the SAS programming APIs to the Java Authentication and Authorization Service (JAAS). For information on this migration, see “Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS)” on page 57.
System Administration Features	
Deprecation	**Recommended Migration Action**
Configuring resources under cell scope is deprecated.	You should configure resources under cluster scope instead. In previous releases, you configured cell scope resources to allow the cluster members to share the resource configuration definition. In Version 6, cell scope resource configuration is discouraged because cell scope resources are visible to every node in the cell, even though not every node in the cell is able to support the resource.
The depl.extension.reg and installdir options for the install command in the AdminApp scripting object are deprecated.	There is no replacement for the depl.extension.reg option. In Version 5.x, this option was a no-op. For the installdir option, use the installed.ear.destination option instead.
Performance Features	
Deprecation	**Recommended Migration Action**
The PMI Client API, which was introduced in Version 4.0 to programmatically gather performance data from WebSphere Application Server, is deprecated.	The Java Management Extension (JMX) interface, which is part of the J2EE specification, is the recommended way to gather WebSphere Application Server performance data. PMI data can be gathered from the J2EE-managed object MBeans, or from the WebSphere PMI Perf MBean. While the J2EE MBeans provide performance data about a specific component, the Perf MBean acts as a gateway to the WebSphere Application Server PMI service, and provides access to the performance data for all the components.

Features deprecated in Version 5.1.1

Application Programming Model and Container Support Features	
Deprecation	**Recommended Migration Action**
The Web services gateway customization API is deprecated.	Plan over time to replace your existing filters with a combination of JAX-RPC handlers and service integration bus mediations.
Application Services Features	
Deprecation	**Recommended Migration Action**
The following Java Database Connectivity (JDBC) drivers are deprecated: • MS SQL Server 2000 Driver for JDBC • SequeLink JDBC driver for MS SQL Server	If you are using either of these JDBC drivers and still want to use MS SQL Server as your database, switch to the Connect JDBC driver. You can purchase the Connect JDBC driver from DataDirect Technologies; or you can use the Connect JDBC driver that is shipped with WebSphere Application Server, which is free for use with WebSphere Application Server.

Features deprecated in Version 5.1

Installation and Migration Tools	
Deprecation	**Recommended Migration Action**
The Application Assembly Tool (AAT) that is used for developing J2EE applications is replaced with the Assembly Tool (ATk) component of the Application Server Toolkit (ASTk).	Instead of running the Application Assembly Tool, users will install and run the Assembly Toolkit component of the Application Server Toolkit. ASTk is based on the Eclipse framework. Upon starting the ASTk, the J2EE function is found by opening the J2EE Perspective.
JDOM (a Java representation of an XML document that provides an API for efficient reading, manipulating, and writing documentation) is deprecated. The currently packaged version of JDOM in WebSphere Application Server will not be packaged in future releases.	Go to JDOM and get the latest copy of JDOM and bundle it inside your application. **Note:** Customers running WebSphere Studio Application Developer Integration Edition Version 4.1 applications will need to migrate them to WebSphere Studio Application Developer Integration Edition Version 5.0.

The C++ Object Request Broker (ORB), the C++ library for IDL value types and the WebSphere Application Server C++ security client are deprecated. IBM will no longer ship or support the Common Object Request Broker Architecture (CORBA) C++ Developer Kit. The CORBA technology is a bridge for migration to a Java 2 Platform Enterprise Edition (J2EE) and WebSphere Application Server environment. In addition to the preceding information, the CORBA C++ client feature will be removed from the Application Clients installation image in future releases.	It is recommended that customers migrate to the Object Request Broker (ORB) service for Java technology that ships with WebSphere Application Server. However, there is no equivalent J2EE functionality for the C++ security client or the C++ value-type library. Customers that require such functionality must provide or develop their own. The deprecation of the CORBA C++ Developer Kit does not affect support for CORBA interoperability with vendor software for CORBA services. View the following links for additional information about interoperability: • CORBA Interoperability Samples documentation • IBM WebSphere Application Servers CORBA Interoperability white paper
IBM Cloudscape Version 5.1.x is deprecated.	Use the Cloudscape Network Server JDBC driver.
Servers and Clustering Features	
Deprecation	**Recommended Migration Action**
IBM HTTP Server (IHS) Version 1.3.x is deprecated.	If you are using IHS Version 1.3.x with modules that: • are shipped as part of IHS Version 1.3.x packages, you do not need to take any action to migrate those modules. • are supplied by a third party (including other IBM products), you need to obtain IHS/Apache 2 versions of these modules from the third party. • have been customized or are inhouse, you need to port these modules to the new IHS/Apache 2 API.
Application Programming Model and Container Support Features	
Deprecation	**Recommended Migration Action**
Bean Scripting Framework (BSF) JavaServer Pages (JSP) execution and debugging functionality is deprecated.	The functionality will need to be rearchitected if you are using the JavaScript, Tcl, and Python languages. If using BSF scripting in your own custom applications, they will be unaffected. Custom scripts written for the WebSphere Application Server administrative console will also be unaffected. This functionality will continue to exist in WebSphere Application Server Version 5.1 and succeeding releases until Version 6.0. If debugging JSP files, you might have to restart the application server during JavaScript debugging sessions.

Data access programming interfaces in com.ibm.websphere.rsadapter.
 	
Relational resource adapter interface: (com.ibm.websphere.rsadapter).
 	
Methods have been deprecated in these types:
`com.ibm.websphere.rsadapter.OracleDataStoreHelper`
`public void doSpecialBLobWork(ResultSet rset, InputStream[] data, String[] blobColumnNames)`
`public String assembleSqlString(String[] blobColumnNames, StringBuffer whereClause, String[] varValues, String tableName)`
	These relational resource adapter deprecated methods do not impact the application.
	Note: You will not need to implement these deprecated methods in their subclasses if you have the subclass of OracleDataStoreHelper class. Those deprecated methods will not be called by the WebSphere Application Server runtime.

Web container API modifications:
Note: There are no declared deprecations. The only changes are caused because of a Java API that changed between 1.3 and 1.4.
 	
The changed class is com.ibm.websphere.servlet.error.ServletErrorReport. The return signature for getStackTrace( ) is changed because java.lang.Throwable now defines the same method with a different return signature.
• Old method signature
`public String getStackTrace();`
`// returns a String representation of the exception stack`
• New method signature (JDK 1.4, WebSphere Application Server 5.1)
`public StackTraceElement[] getStackTrace();`
`// returns an array of stack trace elements`
• Replacement method (WebSphere Application Server 5.1) (a replacement method that carries on the old functionality has been provided):
`public String getStackTraceAsString();`
`// returns a String representation of the Exception Stack`
	If you are using com.ibm.websphere.servlet.error.ServletErrorReport.getStackTrace( ) and expecting a return type of String, you need to change your application to use the replacement method.

Application Services Features	
Deprecation	**Recommended Migration Action**
Data access binaries -- Common Connector Framework: The following JAR files are deprecated: • ccf.jar • ccf2.jar • recjava.jar • eablib.jar	The J2EE Connector Architecture solution should be used instead of the Common Connector Framework.
Setting the XA partner log directory using the TRANLOG_ROOT variable is deprecated.	The setting currently stored in the TRANLOG_ROOT variable (if any) will need to be added to the Transaction Service panel for all servers that need to use the XA partner log. If the default location is to be used, then no action is required. The Transaction Service panel can be found on the administrative console by selecting Application Servers on the left, choosing the application server to be modified, and selecting Transaction Service on the panel that is displayed. The directory currently in TRANLOG_ROOT should be entered in the Logging Directory box on the panel.
Security Features	
Deprecation	**Recommended Migration Action**
The API is deprecated for com.ibm.websphere.security.auth.WSPrincipal.getCredential().	Instead of getting the WSCredential from the principal, you should now use one of the following methods to get the Subject that contains the WSCredential: • The RunAs Subject is the Subject used for outbound requests. • The Caller subject is the Subject that represents the authenticated caller for the current request. • The methods to use to get the runAs and caller subjects are as follows: com.ibm.websphere.security.auth.WSSubject.getRunAsSubject() and com.ibm.websphere.security.auth.WSSubject.getCallerSubject() respectively.
Security programming interface deprecations: • The interface is deprecated in com.ibm.websphere.security.auth.WSSecurityContext. • The exception is deprecated in com.ibm.websphere.security.auth.WSSecurityContextException. • The class is deprecated in com.ibm.websphere.security.auth.WSSecurityContextResult.	Use Java Authentication and Authorization Service (JAAS) for all authentication related functionality.

The Integrated Cryptographic Services Facility (ICSF) authentication mechanism is deprecated in Version 5.1.	Use the Lightweight Third Party Authentication (LTPA) mechanism.
System Administration Features	
Deprecation	**Recommended Migration Action**
The following class is deprecated: com.ibm.websphere.rsadapter.DB2390DataStoreHelper	If you currently use the DB2390DataStoreHelper class for the DB2 Legacy CLI-based provider when you are accessing data, you should now use the DB2DataStoreHelper class. If you currently use the DB2390DataStoreHelper class for the DB2 Universal JDBC provider when you are accessing data, you should now use the DB2UniversalDataStoreHelper class.

Features deprecated in Version 5.0.2

Application Programming Model and Container Support Features	
Deprecation	**Recommended Migration Action**
Apache SOAP channel in Web services gateway.	Gateway services should be deployed to the SOAP HTTP channel instead of the Apache SOAP channel. The Endpoint (URL) of the service will be different for this channel and therefore client programs that are talking to the gateway will need to use the new service Endpoint.
Apache SOAP, WEBSJAVA.SOAP: • soap.jar • wssoap.jar	See the information center for more information.
Application Services Features	
Deprecation	**Recommended Migration Action**

Data access programming interfaces in com.ibm.websphere.rsadapter.
 	
Relational resource adapter interface (com.ibm.websphere.rsadapter)
 	
Methods have been deprecated in these types:
`com.ibm.websphere.rsadapter.DataStoreHelper`
`public int processSQL(java.lang.String.sqlString, int isolevel, boolean addForUpdate, boolean addextendedForUpdateSyntax);`
`public DataStoreAdatperException mapException(DataStoreAdapterException e);`
`com.ibm.websphere.rsadapter.GenericDataStoreHelper`
`public int processSQL(java.lang.String.sqlString, int isolevel, boolean addForUpdate, boolean addextendedForUpdateSyntax);`
`public DataStoreAdatperException mapException(DataStoreAdapterException e);`
`com.ibm.websphere.rsadapter.WSCallHelper`
`public static DataStoreHelper createDataStoreHelper(String dsClassName)`
	These relational resource adapter deprecated methods do not impact the application.
	Note: You will not need to implement these deprecated methods in their subclasses if you have the subclass of GenericDataStoreHelper. Those deprecated methods will not be called by the WebSphere Application Server runtime.
	
	For com.ibm.websphere.rsadapter.WSCallHelper, use the getDataStoreHelper(datasource) method to get a DataStoreHelper object.

System Administration Features	
Deprecation	**Recommended Migration Action**
The **testConnection** command in the AdminControl scripting object ($AdminControl TestConnection configId props) is deprecated. Running this command in WebSphere Application Server, Version 5.0.2 or later returns the following message: `WASX7390E: Operation not supported - testConnection command with config id and properties arguments is not supported. Use testConnection command with config id argument only.`	As of WebSphere Application Server, Version 5.0.2 or later, the preferred way to test a data source connection is the testConnection command passing in the data source configuration ID as the only parameter.
The **getPropertiesForDataSource** command in the AdminControl scripting object (`$AdminControl getPropertiesForDataSource configId`) is deprecated. This command incorrectly assumes the availability of the configuration service when you run it in the connected mode. Running this command in WebSphere Application Server, Version 5.0.2 or later returns the following message: `WASX7389E: Operation not supported - getPropertiesForDataSource command is not supported.`	There is no replacement for this command.

Features deprecated in Version 5.0.1

Application Services Features	
Deprecation	**Recommended Migration Action**

Data access programming interfaces in com.ibm.websphere.rsadapter.
 	
Relational resource adapter interface (com.ibm.websphere.rsadapter).
 	
Methods have been deprecated in these types:
`com.ibm.websphere.rsadapter.DataStoreHelper`
`public int processSQL(java.lang.String sqlString, int isolevel);`
`com.ibm.websphere.rsadapter.GenericDataStoreHelper`
`public int processSQL(java.lang.String sqlString, int isolevel);`
`com.ibm.websphere.rsadapter.DB2390DataStoreHelper`
`public int processSQL(java.lang.String sqlString, int isolevel);`
	These relational resource adapter deprecated methods do not impact the application.
	Note: You will not need to implement these deprecated methods in their subclasses if you have the subclass of com.ibm.websphere.rsadapter.GenericDataStoreHelper. Those deprecated methods will not be called by the WebSphere Application Server runtime.

Features deprecated in Version 5.0

Application Services Features	
Deprecation	**Recommended Migration Action**

The following three methods from com.ibm.websphere.appprofile.accessintent.AccessIntent are deprecated:
`public boolean getPessimisticUpdateHintWeakestLockAtLoad();`
`public boolean getPessimisticUpdateHintNoCollision();`
`public boolean getPessimisticUpdateHintExclusive();`
 	
This is a base API.
	Rather than using the three deprecated methods on the AccessIntent interface, developers should use the following method from the same interface:
	`public int getPessimisticUpdateLockHint();`
	
	The possible return values are defined on the AccessIntent interface:
	`public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NOCOLLISION = 1;`
	`public final static int PESSIMISTIC_UPDATE_LOCK_HINT_WEAKEST_LOCK_AT_LOAD = 2;`
	`public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NONE = 3;`
	`public final static int PESSIMISTIC_UPDATE_LOCK_HINT_EXCLUSIVE = 4;`

Web application programming interfaces -- Various Version 5.x methods in com.ibm.websphere.ServletErrorReport	
Security Features	
Deprecation	**Recommended Migration Action**
The com.ibm.websphere.security.CustomRegistry interface is deprecated.	Use the com.ibm.websphere.security.UserRegistry interface. See “Migrating custom user registries” on page 52.
Performance Features	
Deprecation	**Recommended Migration Action**
Performance Monitoring Infrastructure -- Various Version 5.x public methods in: • com.ibm.websphere.pmi.stat.StatsUtil • com.ibm.websphere.pmi.PmiJmxTest • com.ibm.websphere.pmi.client.PmiClient	These methods are replaced as follows: • com.ibm.websphere.pmi.stat.StatsUtil There is no replacement for StatsUtil. • com.ibm.websphere.pmi.PmiJmxTest Use PmiClient.findConfig() instead. • com.ibm.websphere.pmi.client.PmiClient The getNLSValue (String key) is replaced with getNLSValue (String key, String moduleID).

Removal list

The following tables describe what is removed—such as features, APIs, scripting interfaces, tools, wizards, publicly exposed configuration data, naming identifiers, and constants. Where possible, the recommended replacement is identified.

- “Features removed in Version 6.1” on page 12

• “Features removed in Version 6.0” on page 13

Features removed in Version 6.1

Feature	Recommended Migration Action
com.ibm.websphere.security.CustomRegistry interface	Use the com.ibm.websphere.security.UserRegistry interface. See “Migrating custom user registries” on page 52.
Support for the Secure Authentication Service (SAS) IIOP security protocol	Use the Common Secure Interoperability Version 2 (CSIv2) protocols.
Support for the Secure Authentication Service (SAS) CORBA security programming APIs	Migrate from the SAS programming APIs to the Java Authentication and Authorization Service (JAAS). For information on this migration, see “Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS)” on page 57.
Support for the Common Connector Framework (CCF)	Use the J2EE Connector Architecture (JCA) solution.
Support for the IBM Cloudscape Version 5.1.x database	Use the IBM Cloudscape Version 10.1 database. This database provides Derby Version 10.1 binaries, NLS enablement, QA, and IBM problem support. **Note:** When using WebSphere Application Server, you will see the term ″Derby″ rather than ″Cloudscape″ used in places such as the administrative console, the ejbdeploy command, and others.
Log Analyzer, the tool that was previously provided for viewing and analyzing the activity or service log file	Use the Log and Trace Analyzer tool for Eclipse in the Application Server Toolkit (AST). This tool is installable from the AST launchpad console. For more information, go to **Application Server Toolkit > Detecting and analyzing runtime problems > Log and Trace Analyzer** in this information center.
Mozilla Rhino JavaScript (js.jar)	Use the Rhino code available from Mozilla. Go to Rhino: JavaScript for Java and get the latest copy of Rhino.
Java Document Object Model (JDOM)	Use the code available from the JDOM organization. Go to JDOM, get the latest copy of JDOM, and bundle it inside your application.
Class preloading function	No migration action is necessary.
The following samples from the Samples Gallery: • Adventure Builder • Greenhouse by WebSphere • WebSphere Bank The following technology samples from the Samples Gallery: • Bean-Managed Persistence (BMP) • Container-Managed Persistence (CMP) 1.1 • Container-Managed Persistence (CMP) 2.1 • Container-Managed Relationships (CMR) • EJB Time • Filter Servlet • JavaServer Pages (JSP) 2.0 • Message-Driven Beans (MDB) • Pagelist Servlet • Simple JavaServer Pages (JSP) • Simple Servlet • Stateful Session • TagLib	No migration action is necessary.

Features removed in Version 6.0

Component	Classes and interfaces
activity	com.ibm.ws.activity.ActivityConstants com.ibm.ws.activity.ActivityService com.ibm.ws.activity.ActivityServiceInitializer com.ibm.ws.activity.ActivityTrace com.ibm.ws.activity.GlobalIdImpl com.ibm.ws.activity.HighlyAvailableServiceManager com.ibm.ws.activity.HLSLiteDataInterface com.ibm.ws.activity.HLSLiteExtended com.ibm.ws.activity.HLSLiteInfo com.ibm.ws.activity.j2ee_activity_specific_data com.ibm.ws.activity.j2ee_activity_specific_dataHelper com.ibm.ws.activity.ServiceMigration com.ibm.ws.activity.VUTrace com.ibm.ws.activity.WebSphereServiceManager com.ibm.ws.activity.WebSphereUserActivity com.ibm.ws.javax.activity.ActionErrorException com.ibm.ws.javax.activity.ActionNotFoundException com.ibm.ws.javax.activity.ActivityCoordinator com.ibm.ws.javax.activity.ActivityInformation com.ibm.ws.javax.activity.ActivityManager com.ibm.ws.javax.activity.ActivityNotProcessedException com.ibm.ws.javax.activity.ActivityPendingException com.ibm.ws.javax.activity.ActivityToken com.ibm.ws.javax.activity.CompletionStatus com.ibm.ws.javax.activity.ContextPendingException com.ibm.ws.javax.activity.CoordinationInformation com.ibm.ws.javax.activity.GlobalId com.ibm.ws.javax.activity.InvalidParentContextException com.ibm.ws.javax.activity.InvalidStateException com.ibm.ws.javax.activity.NoActivityException com.ibm.ws.javax.activity.NoImplementException com.ibm.ws.javax.activity.NotOriginatorException com.ibm.ws.javax.activity.Outcome com.ibm.ws.javax.activity.PersistentActivityCoordinator com.ibm.ws.javax.activity.PropertyGroupContext com.ibm.ws.javax.activity.PropertyGroupRegisteredException com.ibm.ws.javax.activity.PropertyGroupUnknownException com.ibm.ws.javax.activity.ServiceAlreadyRegisteredException com.ibm.ws.javax.activity.ServiceInformation com.ibm.ws.javax.activity.ServiceNotRegisteredException com.ibm.ws.javax.activity.Signal com.ibm.ws.javax.activity.SignalSetActiveException com.ibm.ws.javax.activity.SignalSetInactiveException com.ibm.ws.javax.activity.SignalSetUnknownException com.ibm.ws.javax.activity.Status com.ibm.ws.javax.activity.SystemException com.ibm.ws.javax.activity.TimeoutRangeException com.ibm.ws.javax.activity.UserActivity com.ibm.ws.javax.activity.coordination.Action com.ibm.ws.javax.activity.coordination.RecoverableAction com.ibm.ws.javax.activity.coordination.ServiceManager com.ibm.ws.javax.activity.coordination.SignalSet com.ibm.ws.javax.activity.coordination.SubordinateSignalSet com.ibm.ws.javax.activity.propertygroup.PropertyGroup com.ibm.ws.javax.activity.propertygroup.PropertyGroupManager com.ibm.ws.javax.ejb.ActivityCompletedLocalException com.ibm.ws.javax.ejb.ActivityRequiredLocalException com.ibm.ws.javax.ejb.InvalidActivityLocalException
admin	com.ibm.websphere.management.application.EarUtils
als	com.ibm.websphere.als.BufferManager
anttasks	com.ibm.websphere.ant.tasks.endptEnabler.Property com.ibm.websphere.ant.tasks.Java2WSDL.Mapping com.ibm.websphere.ant.tasks.Messages com.ibm.websphere.ant.tasks.WSDL2Java.Mapping
dynacache	com.ibm.websphere.servlet.cache.CacheConfig

Component	Classes and interfaces
ras	com.ibm.ras.RASConsoleHandler com.ibm.ras.RASEnhancedMessageFormatter com.ibm.ras.RASEnhancedTraceFormatter com.ibm.ras.RASErrorHandler com.ibm.ras.RASFileHandler com.ibm.ras.RASFormatter com.ibm.ras.RASHandler com.ibm.ras.RASMessageFormatter com.ibm.ras.RASMultiFileHandler com.ibm.ras.RASSerialFileHandler com.ibm.ras.RASSocketHandler com.ibm.ras.RASTextAreaHandler com.ibm.ras.RASTraceFormatter com.ibm.websphere.ras.WsOrbRasManager
security	com.ibm.websphere.security.AuthorizationTable com.ibm.websphere.security.FileRegistrySample com.ibm.websphere.security.SecurityProviderException com.ibm.websphere.security.WASPrincipal com.ibm.websphere.security.auth.AuthDataFileEnc
userprofile	com.ibm.websphere.userprofile.UserProfile com.ibm.websphere.userprofile.UserProfileCreateException com.ibm.websphere.userprofile.UserProfileExtender com.ibm.websphere.userprofile.UserProfileFinderException com.ibm.websphere.userprofile.UserProfileManager com.ibm.websphere.userprofile.UserProfileProperties com.ibm.websphere.userprofile.UserProfileRemoveException
Scheduler API	com.ibm.websphere.scheduler.pmi.SchedulerPmiModule com.ibm.websphere.scheduler.pmi.SchedulerPerf
Scheduler API	com.ibm.websphere.scheduler.MessageTaskInfo.setJMSPriority()
ObjectPool APIs	com/ibm/websphere/objectpool/pmi/ObjectPoolPerf.java com/ibm/websphere/objectpool/pmi/ObjectPoolPmiModule.java
Asynchronous Beans APIs	com/ibm/websphere/asynchbeans/pmi/AlarmManagerPerf.java com/ibm/websphere/asynchbeans/pmi/AsynchBeanPerf.java com/ibm/websphere/asynchbeans/pmi/SubsystemMonitorManagerPerf.java com/ibm/websphere/asynchbeans/pmi/SubsystemMonitorPerf.java com/ibm/websphere/asynchbeans/pmi/AlarmManagerPmiModule.java com/ibm/websphere/asynchbeans/pmi/AsynchBeanPmiModule.java com/ibm/websphere/asynchbeans/pmi/SubsystemMonitorManagerPmiModule.java com/ibm/websphere/asynchbeans/pmi/SubsystemMonitorPmiModule.java

Chapter 2. How do I migrate, coexist, and interoperate?

Use the documentation provided to answer your questions about migration.

Hold your cursor over the task icon () to see a description of the task. The task preview feature is unavailable for Mozilla Web browsers.

Obtain an overview of migration and coexistence options	Documentation
Determine what configuration information needs to change	Documentation
Locate the migration tools	Documentation
Migrate Web server configurations	Documentation
Identify deprecated features that you might be using	Documentation
Review the software and hardware prerequisites	Documentation
Review supported coexistence scenarios	Documentation
Obtain valid port settings for coexistence	Documentation
Interoperate across product versions	Documentation

Chapter 3. Migrating applications

Use this section to learn about migrating applications.

Web applications

Migrating to Java 2 Standard Edition (J2SE) 5

This product version supports the Java 2 Standard Edition (J2SE) 5 specification. Its Java virtual machine provides a Java language compiler and execution environment. Decide whether your new and existing applications will take advantage of the capabilities added by J2SE 5, adjust the JIT mode if necessary, and begin the transition from deprecated functions.

For an introduction to J2SE 5, see the ″J2SE 5 in a Nutshell″ article on the Sun site at http://java.sun.com/developer/technicalArticles/releases/j2se15/. The following JSRs are new in J2SE 5:

- JSR 003: The JMX 1.2 specification. Packages: javax.management.*
- JSR 013: Additions to java.math for improved arithmetic operations using BigDecimal
- JSR 028: The Java SASL packages: javax.security.sasl
- JSR 114: JDBC Rowset implementations that specify rowset more completely
- JSR 133: Java Memory Model and Thread Specification Revision
- JSR 160: JMX remote API, V1.0
- JSR 163: Java Platform Profiling Architecture, JVMTI (replaces JVMPI)
- JSR 166: Concurrency utilities. Packages: java.util.concurrent.*
- JSR 174: Monitoring and Management Specification for the Java Virtual Machine
- JSR 175: A Metadata Facility for the Java Programming Language
- JSR 200: Network Transfer Format for Java Archives
- JSR 201: Extending the Java Programming Language with Enumerations, Autoboxing, Enhanced for Loops and Static Import
- JSR 206: Java API for XML Processing (JAXP) 1.3
- JSR 204: Unicode Supplementary Character Support

The new virtual machine specification adds several features and functions to benefit application developers, such as generics, auto-boxing of primitives, annotations (API documentation metadata for code generation), and support for enumerated types. This makes development quicker, easier, and less error prone. For example, generics should help eliminate issues with ClassCastExceptions from items like vectors, as generics based containers will allow compile-time catching of incorrect assignment or casting. (For developers familiar with the C++ language, generics are a new Java language function similar to C++ templates.)

For details, see the J2SE 5 application programming interface documentation on the Sun site at http://java.sun.com/j2se/1.5.0/docs/api/index.html. See also http://java.sun.com/j2se/1.5.0/docs/index.html.

- Decide whether to take advantage of new J2SE 5 capabilities in your applications.

  Applications using the new language features and J2SE 5 can be deployed only to Version 6.1 nodes, as earlier product versions do not provide the J2SE 5 virtual machine.

  The J2EE 1.4 specification does not take into account the new language features. Therefore, the usage of generics based types should not be used with public EJB interfaces that are exposed on the home, stubs, and so forth.

If the code being developed must run on multiple J2SE levels, use only the API specification for the minimum J2SE level, such as J2SE 1.4, to avoid inadvertent usage of classes and methods that are not part of all of the required J2SE levels. Failure to do so may cause application breakage on older J2SE implementations.

Applications that access classes and APIs internal to the Java virtual machine could have problems. These classes and APIs are not covered by the J2SE 5 specification and are therefore subject to change. Packages with prefixes such as 'com.sun.*' are considered internal. Additionally, direct use of implementations of XML and XSL parsers is strongly discouraged, such as direct use of Xerces and Xalan classes that provide the JAXP implementation for the virtual machine. The direct parser APIs also are considered internal and subject to change. Applications should rely only on the JAXP APIs defined in the J2SE 1.4 and J2SE 5 API documentation. If your application requires a specific version of Xerces or Xalan, or some other XML/XSL parser package, then embed the parser within your application's `WEB-INF/lib` directory and set the appropriate class loading mode in your application deployment so that for your application the XML parser APIs are loaded from the application class path, not the Java virtual machine bootstrap class path. Failure to follow this guideline can cause significant problems when trying to migrate to a new J2SE level.

- Compile J2SE 5 applications to run on older Java virtual machine levels by setting the compiler modes.

  When compiling applications that are built with J2SE 5 that are intended for running on older J2SE specifications, be sure to specify '-source' and '-target' modes for the J2SE 5 compiler. Doing so ensures that the bytecode generated is compatible with the earlier Java virtual machine.

  For example, if the target Java virtual machine is at 1.4.2 level, when you compile applications with J2SE 5, you should specify '-source 1.4', and 'target 1.4' to generate bytecode compatible with 1.4.2. This does not handle the usage of packages, classes, or functions new to J2SE 5. It only addresses bytecode output. Developers must take care in what APIs they are using from the J2SE packages if they intend to run the application on multiple Java virtual machine specification levels.

- Address incompatibilities in previously compiled J2SE 1.4 based applications.

  Java (TM) 2 Technology Edition, Version 5.0 is upwards binary-compatible with Java (TM) 2 Technology Edition, Version 1.4.2, except for the incompatibilities documented by Sun Microsystems at http://java.sun.com/j2se/1.5.0/compatibility.html#binary. Most of the incompatibilities refer to compiling classes at a JDK 5 level using a target of 1.5. ″Almost all existing programs should run on J2SE 5.0 without modification,″ to cite the Sun documentation. As a migration reference, the J2SE 1.4 and J2SE 5 application programming interface documentation is available on the Sun site at http://java.sun.com/j2se/1.4.2/docs/api/index.html and http://java.sun.com/j2se/1.5.2/docs/api/index.html, respectively.

  Here are the most notable source compatibility problems.

  – **Variables named 'enum.'** The word 'enum' has become a language keyword. It now will cause a compiler fault if used as a variable name. Consider specifying '-source 1.4' for compilations with J2SE 5 until you can correct the variable names. While using -source 1.4, all new JDK 5 language constructs are disabled and cannot be used in the source code.

  – **Ambiguous references to classes with base names of 'Proxy,' 'Queue,' or 'Formatter.'** You might encounter compile-time errors if you import java.net.* and then use other classes with a base name of Proxy without fully qualifying the latter class names. The errors are because java.net.* now contains java.net.Proxy.

    In similar circumstances, you might encounter errors importing java.lang.reflect.* Note also that a new java.utils.Queue class in J2SE 5 conflicts with other Queue package names, such as javax.jms.Queue.

    You might encounter compile-time errors if you import java.util.* and then use other classes with a base name of Formatter, without fully qualifying the latter class names. The errors occur because java.util.* now contains java.util.Formatter, a class added in the J2SE 5 spec.

- Start the transition from deprecated JVMDI and JVMPI functions to JVMTI.

  J2SE 5 deprecates some functions that were available for public usage in previous J2SE specifications. JVMDI and JVMPI are deprecated in J2SE 5 and might be removed in the next major release of the J2SE specification. Any new development and tool sets should begin moving to JVMTI. Migrate any of

your native (JNI) performance profiling libraries to the new JVMTI API described at http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/index.html.

- Update your use of the Java command line interface.

  The command-line interfaces for the J2SE 5 level have not changed extensively from J2SE 1.4, although they vary among virtual machine vendors. You can find them in the `JAVA_HOME/bin` directory. Here are some notable command line options that are standard to all J2SE 5 implementations.

  – For JVMTI, use `-agentlib` to load a native agent library that you specify.
  – For JVMTI, use `-agentpath` to load the native agent library by the full path name
  – For JVMTI, use `-javaagent` to load the Java programming language agent (see java.lang.instrument for details)
  – See `apt -help` for information about this new command line supporting the annotations capability.
  – See `javac -help` for information and updates to that command line.

- Update ANT tasks.

  If you have created ANT tasks based on the idltojava ANT task shipped with prior versions of this product, you will need to ensure that it passes the proper parameters for J2SE 5 as it does for J2SE 1.4, to ensure the stubs/ties and skeletons it generates are compatible to earlier product releases.

JavaServer Pages migration best practices and considerations

The standard JavaServer Pages (JSP) tags from JSP 1.1 such as jsp:include, jsp:useBean, and <%@ page %>, a will migrate successfully to JSP 2.0. However, there are several areas that must be considered when migrating JavaServer Pages. This topic discusses the areas that you must consider when migrating JavaServer Pages.

Classes from the unnamed or default package

As of JSP 2.0, referring to any classes from the unnamed or default package is not allowed. This can result in a translation error on some containers, specifically those that run in a JDK 1.4 or greater environment which will also break compatibility with some older JSP applications. However, as of JDK 1.4, importing classes from the unnamed package is not valid. See Java 2 Platform, Standard Edition Version 1.4.2 Compatibility with Previous Releases for details. Therefore, for forwards compatibility, applications must not rely on the unnamed package. This restriction also applies for all other cases where classes are referenced, such as when specifying the class name for a tag in a Tag Library Descriptor (TLD) file.

Page encoding for JSP documents

There have been noticeable differences in internationalization behavior on some containers as a result of ambiguity in the JSP 1.2 specification. However, steps were taken to minimize the impact on backwards compatibility and overall, the internationalization abilities of JSP files have been greatly improved.

In JSP specification versions prior to JSP 2.0, JSP pages in XML syntax, JSP documents, and those in standard syntax determined their page encoding in the same fashion, by examining the pageEncoding or contentType attributes of their page directive, defaulting to ISO-8859-1 if neither was present.

As of JSP 2.0, the page encoding for JSP documents is determined as described in section 4.3.3 and appendix F.1 of the XML specification, and the pageEncoding attribute of those pages is only checked to make sure it is consistent with the page encoding determined as per the XML specification. As a result of this change, JSP documents that rely on their page encoding to be determined from their pageEncoding attribute are no longer decoded correctly. These JSP documents must be changed to include an appropriate XML encoding declaration.

Additionally, in JSP 1.2, page encodings are determined on translation unit basis whereas in JSP 2.0, page encodings are determined on the basis of each file. Therefore, if the a.jsp file statically includes the b.jsp file, and a page encoding is specified in the a.jsp file but not in the b.jsp file, in JSP 1.2 the encoding

for the a.jsp file is used for the b.jsp file, but in JSP 2.0, the default encoding is used for the b.jsp file.

web.xml file version

The JSP container uses the version of the web.xml file to determine whether you are running a JSP 1.2 application or a JSP 2.0 application. Various features can behave differently depending on the version of the web.xml file. The following is a list of things JSP developers should be aware of when upgrading their web.xml file from version Servlet 2.3 to version Servlet 2.4:

1. EL expressions are ignored by default in JSP 1.2 applications. When you upgrade a Web application to JSP 2.0, EL expressions are interpreted by default. You can use the escape sequence \$ to escape EL expressions that should not be interpreted by the container. Alternatively, you can use the isELIgnored page directive attribute, or the <el-ignored> configuration element to deactivate EL for entire translation units. Users of JSTL 1.0 must upgrade their taglib imports to the JSTL 1.1 uris or use the _rt versions of the tags, for example, use c_rt instead of c or fmt_rt instead of fmt.
2. Web applications that contain files with an extension of .jspx will have those files interpreted as JSP documents, by default. You can use the JSP configuration element <is-xml> to treat .jspx files as regular JSP pages, but there is no way to disassociate .jspx from the JSP container.
3. The escape sequence \$ was not reserved in JSP 1.2. The output for any template text or attribute value that appeared as \$ in JSP 1.2 was \$, however, the output now is just $.

jsp:useBean tag

WebSphere Application Server version 5.1 and later enforces more strict adherence to the specification for the jsp:useBean tag: with type and class attributes. Specifically, you should use the type attribute should be used to specify a Java type that cannot be instantiated as a JavaBean. For example, a Java type that is an abstract class, interface, or a class with no public no-args constructor. If the class attribute is used for a Java type that cannot be instantiated as a JavaBean, the WebSphere Application Server JSP container produces a unrecoverable translation error at translation time.

Generated packages for JSP classes

Any reliance on generated packages for JSP classes will result in non-portable JSP files. Packages for generated classes are implementation-specific and therefore you should not rely on these packages.

JspServlet class

Any reliance on the existence of a JspServlet class will cause unrecoverable error problems. WebSphere Application Server version 6.0 and later no longer uses a JspServlet class.

JavaServer Faces

JavaServer Faces (JSF) is a user interface framework or API that eases the development of Java based Web applications. WebSphere Application Server version 6.1 supports JavaServer Faces 1.1 at a runtime level, therefore using JSF reduces the size of the Web application since runtime binaries no longer need to be included in your Web application.

The JSF runtime also :

- Makes it easy to construct a user interface from a set of reusable user interface components
- Simplifies migration of application data to and from the user interface
- Helps manage user interface state across server requests
- Provides a simple model for wiring client-generated events to server-side application code
- Allows custom user interface components to be easily build and reused

The Sun JSF Reference Implementation provides the foundation of the code used for the JSF support in WebSphere Application Server. However, some dependencies on Jakarta APIs have been removed and replaced with Application Server specific solutions as a result of potential problems that may occur when Open Source APIs are included in the Application Server runtime. For example, when included in the Application Server runtime, these Open Source APIs are made available to all applications installed within the Application Server, therefore bringing versioning, support and legal issues. The version of the JSF runtime provided by the Application Server resides in the normal runtime library location and is available to all Web applications that leverage JSF APIs. The loading of the JSF servlet works in the same manner as if the runtime was packaged with the Web application.

The following open source dependencies are replaced with other APIs or in-house versions:

- Jakarta Commons BeanUtils
- Jakarta Commons Collections
- Jakarta Commons Digester
- Jakarta Commons Logging
- Mozilla Assert API

The JSF Specification requires JavaServer Pages Standard Tag Library (JSTL) as a dependency, therefore the required version of the JSTL from Jakarta is made available in the Application Server runtime.

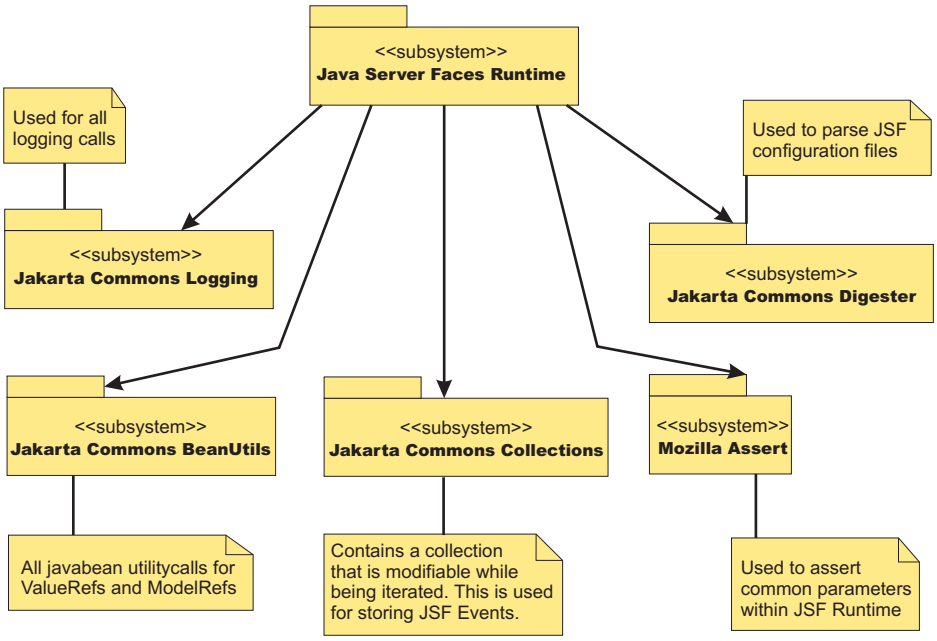
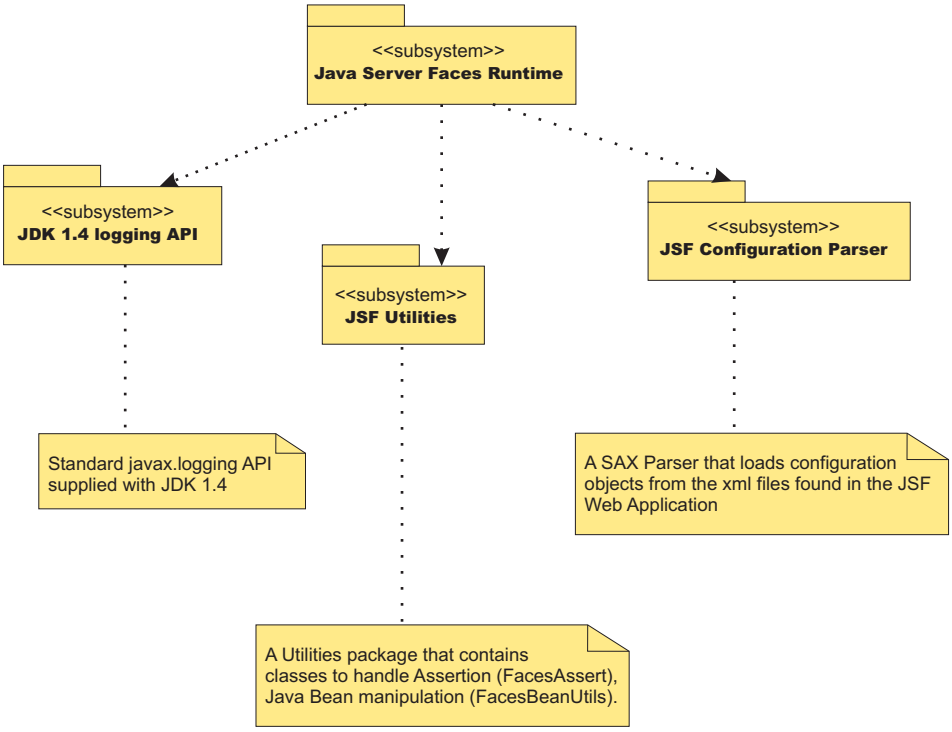


Figure 1. Current external API dependencies from the Sun based JSF runtime

Figure 2. Replacement APIs



The specification related classes (javax.faces.*) for JSF and the IBM modified version of the JSF Sun reference implementation are packaged in the Application Server runtime.

Typically Web applications that leverage this API/Framework embed the JSF API and implementation JAR files within their WAR file. This is not required when these Web applications are deployed and run within WebSphere Application Server. Only the removal of these jars along with any JSTL JAR files from the WAR file is required.

If a Web application requires the use of its own version of JSF or JSTL embedded within it, you can change the class loader mode of the Web application. By default this is set to PARENT_FIRST mode. Changing this value to PARENT_LAST allows the Web application version of the JSF or JSTL classes to load before the WebSphere Application Server.

FacesAssert class

The Sun Reference implementation uses a utility class from Mozilla to perform assertion style calls to method parameters. The faces assert class provides equivalent functionality. The option of leveraging the assertion functionality available in JDK 1.4 is not possible due to the requirement of providing JVM level parameters to turn on assertion code support. The FacesAssert class only contains static method and has no life cycle.

FacesAssert
+ notEmpty ([in] str : String) : boolean
+ nonNull ([in] isNull : Object) : boolean
+ wsAssert ([in] message : String) : boolean
+ wsAssert ([in] argument : boolean , [in] message : String) : boolean

FacesBeanUtils class

The FacesBeanUtils class provides static method replacements for methods used in the Jakarta Commons BeanUtils API. The FacesBeanUtils class has no life cycle.

FacesBeanUtils
+ getProperty ([in] bean : Object , [in] property : String) : Object
+ getPropertyType ([in] bean : Object , [in] property : String) : Class
+ getSimpleProperty ([in] bean : Object, [in] property : String , [in] value : Object)
+ getProperty ([in] bean : Object , [in] property : String , [in] value : Object)
+ convertFromString ([in] value : String, [in] valueClass : Class) : Object
+ convert ([in] targetType : Class , [in] bean : String) : Object

Faces configuration parser

The Sun Reference Implementation of JavaServer Faces use the Jakarta Commons Digester API to parse Faces configuration files. An XML SAX based parser is provided for the Application Server . The Digester code uses reflection code to perform its parsing. This has been found to be quite slow when large configuration files are parsed. The FaceConfigParser class in the diagram below is custom written for the Faces Configuration DTD and therefore parses large configuration files more quickly.

Figure 3. Faces configuration parser

Migrating Web application components from WebSphere Application Server Version 5.x

Migration of Web applications deployed in WebSphere Application Server Version 5.x is not necessary; version 2.2 and 2.3 of the Java Servlet specification and version 1.2 and 1.4 of the JavaServer Pages (JSP) specification are still supported. However, where there are behavioral differences between the Java 2 Enterprise Edition (J2EE) 1.2 and J2EE 1.3 specifications, bear in mind that J2EE 1.3 specifications are implemented in WebSphere Application Server Version 5.x and will override any J2EE 1.2 behaviors.

Servlet migration might be a concern if your application:

- implements a WebSphere Application Server internal servlet to bypass a WebSphere Application Server Version 4.x single application path restriction.
- extends a PageListServlet that relies on configuration information in the servlet configuration XML file.

- calls the `response.sendRedirect` method for a servlet using the `encodeRedirectURL` function or executing within a non-context root.

JSP migration might be a concern if your application references JSP page implementation classes in unnamed packages, or if you install WebSphere Application Server Version 4.x EAR files (deployed in Version 4.x with the JSP `Precompile` option), in Version 5.x. You need to recompile all JSP pages when migrating from WebSphere Application Server Version 5.x.

Follow these steps if migration issues apply to your Web application:

1. If a migrated application references internal servlets, enabled or disabled the functionality through the IBM WebSphere Extensions XMI file, `ibm-web-ext.xmi`, located in each Web module WEB-INF directory or by using assembly tools. Examples of this are fileServing and serveServletsByClassName.
2. Use WebSphere Application Server Version 5.x package names for any WebSphere Application Server Version 4.x internal servlets, which are implemented in your application.

   To bypass the errors, and to enable the serving of static files from the root context, WebSphere Application Server Version 4.x users are advised to open a Web deployment descriptor editor using an assembly tool and select **File serving enabled** on the **Extensions** tab.

HTTP session migration

There are no programmatic changes required to migrate from version 5.x to version 6.x. This article describes features that are available after migration.

Migration from Version 5.x

Note: In Version 5 and later, default write frequency mode is TIME_BASED_WRITES, which is different from Version 4.0.x default mode of END_OF_SERVICE.

When you migrate between releases of WebSphere Application Server Version 5.x and later and you are using a database for session persistence, you can share the session database table between releases. For example, if you are accessing applications that are on WebSphere Application Server version 5.x you can share the session id with applications running on Version 6.x.

EJB applications

Migrating enterprise bean code to the supported specification

Support for Version 2.1 of the Enterprise JavaBeans (EJB) specification is added for Version 6 of this product. Migration of enterprise beans deployed in Versions 4 or 5 of this product is not generally necessary; Versions 1.1 and 2.0 of the EJB specification are still supported.

Follow these steps as appropriate for your application deployment.

1. Modify enterprise bean code for changes in the specification.
   - For Version 1.0 beans, migrate at least to Version 1.1.
   - As stated previously, migration from Version 1.1 to Version 2.x of the EJB specification is not required for redeployment on this version of the product. However, if your application requires the capabilities of Version 2.x, migrate your Version 1.1-compliant code.

     Note: The EJB Version 2.0 specification mandates that prior to the EJB container's running a findBy*Method* query, the state of all enterprise beans enlisted in the current transaction be synchronized with the persistent store. (This is so the query is performed against current data.) If Version 1.1 beans are reassembled into an EJB 2.x-compliant module, the EJB container synchronizes the state of Version 1.1 beans as well as that of Version 2.x beans. As a result, you might notice some change in application behavior even though the application code for the Version 1.1 beans has not been changed.

2. You might have to modify code for some EJB 1.1-compliant modules that were not migrated to Version 2.x. Use the following information to help you decide.
   - Some stub classes for deployed enterprise beans have changed in the Java 2 Software Development Kit, Version 1.4.1.
   - The task of generating deployment code for enterprise beans changed significantly for EJB 1.1-compliant modules relative to EJB 1.0-compliant modules.
3. Reassemble and redeploy all modules to incorporate migrated code.

Migrating enterprise bean code from Version 1.0 to Version 1.1

The following information generally applies to any enterprise bean that currently complies with Version 1.0 of the Enterprise JavaBeans (EJB) specification.

For more information about migrating code for beans produced with Rational Application Developer, see the documentation for that product. For more information about migrating code in general, see ″Resources for learning.″

1. In session beans, replace all uses of javax.jts.UserTransaction with javax.transaction.UserTransaction. Entity beans may no longer use the UserTransaction interface at all.
2. In finder methods for entity beans, include FinderException in the `throws` clause.
3. Remove throws of java.rmi.RemoteException; throw javax.ejb.EJBException instead. However, continue to include RemoteException in the `throws` clause of home and remote interfaces as required by the use of Remote Method Invocation (RMI).
4. Remove uses of the finalize() method.
5. Replace calls to getCallerIdentity() with calls to getCallerPrincipal(). The use of getCallerIdentity() is deprecated.
6. Replace calls to isCallerInRole(Identity) with calls to isCallerinRole (String). The use of isCallerInRole(Identity) and java.security.Identity is deprecated.
7. Replace calls to getEnvironment() in favor of JNDI lookup. As an example, change the following code:

   ```
   String homeName =
      aLink.getEntityContext().getEnvironment().getProperty("TARGET_HOME_NAME");
   if (homeName == null) homeName = "TARGET_HOME_NAME";
   ```

   The updated code would look something like the following:

   ```
   Context env = (Context)(new InitialContext()).lookup("java:comp/env");
   String homeName = (String)env.lookup("ejb10-properties/TARGET_HOME_NAME");
   ```
8. In ejbCreate methods for an entity bean with container-managed persistence (CMP), return the bean's primary key class instead of `void`.
9. Add the getHomeHandle() method to home interfaces.

   ```
   public javax.ejb.HomeHandle getHomeHandle() {return null;}
   ```

Consider enhancements to match the following changes in the specification:
- Primary keys for entity beans can be of type java.lang.String.
- Finder methods for entity beans return java.util.Collection.
- Check the format of any JNDI names being used. Local name spaces are also supported.
- Security is defined by role, and isolation levels are defined at the method level rather than at the bean level.

Enabling security on EJB method group authority:

You might have to enable security on EJB method group authority.

If EJB security is defined for EJB 1.0, an additional step after migration is required to bind security roles to EJB 1.1 method names. Some of the method names for enterprise beans have changed with EJB 1.1. This table provides some examples:

EJB 1.0 method name	EJB 1.1 method name
ejbCreate	create
ejbRemove	remove
ejbGetEJBMetaData	getEJBMetaData
ejbFindBy	findBy

After you migrate your application, use the Application Server Toolkit (AST) to add the EJB 1.1 method name to the method permission created for the EJB 1.0 method. To add the EJB 1.1 method name, first map a network drive from your workstation to the iSeries where the WebSphere Application Server installation resides. Use the ATK to perform these steps.

1. Start the Application Server Toolkit.
   a. On your workstation, select **Start > Programs > IBM > ASTK > ASTK**.
   b. In the Application Server Toolkit window, specify the workspace directory and click **OK** to launch the graphical user interface.
2. In the J2EE perspective, click **File > Import > EAR file** then click **Next**.
   a. In the **EAR file** combination box, click **Browse**.
   b. Locate the /QIBM/Userdata/WebAsAdv4/myinstance/installedApps/ directory on your mapped iSeries drive. Select the EAR file that contains the EJB module that you want to update and click **OK**.
   c. In the Project name field, type a name for the enterprise application project that will be created when you import the EAR file.
   d. In the Project location field, enter the directory where the project source files will be stored.

      By default, the current workspace directory is used. Click **Browse** to choose another location.
   e. If you do not want to be warned about overwriting existing resources, select **Overwrite existing resources without warning**.
   f. Click **Finish** to accept all defaults for EAR import.
3. Expand **EJB Modules > *module_name***, where *module_name* is the name of the module that you want to update.
4. Right-click the desired EJB module, and select **Open With > Deployment Descriptor Editor** from the context menu.
5. On the **EJB Deployment Descriptor** page, click the **Assembly Descriptor** tab.
6. On the **Assembly Descriptor** page, select the method permission that contains the EJB 1.0 method name under the **Method Permissions** heading.
7. Click **Edit**. The **Edit Method Permission**wizard appears.
8. Select the security role for the method permission from the list of roles found.
9. Click **Next**.
10. Select one or more enterprise beans from the list of beans found.
11. Click **Next**.
12. Select the appropriate methods that contain the EJB 1.1 method name that you want to bind to your security role.
13. Unselect the corresponding methods that contain the EJB 1.0 method name.
14. Click **Finish**.
15. Click **File > Export** to save the updated EAR file to a temporary file. Reinstall the application using the temporary file.

Migrating enterprise bean code from Version 1.1 to Version 2.1

Enterprise JavaBeans (EJB) Version 2.1-compliant beans can be assembled only in an EJB 2.1-compliant module, although an EJB 2.1-compliant module can contain a mixture of Version 1.x and Version 2.1 beans.

The EJB Version 2.1 specification mandates that prior to the EJB container starting a findBy*Method* query, the state of all enterprise beans that are enlisted in the current transaction be synchronized with the persistent store. (This action is so the query is performed against current data.) If Version 1.1 beans are reassembled into an EJB 2.1-compliant module, the EJB container synchronizes the state of Version 1.1 beans as well as that of Version 2.1 beans. As a result, you might notice some change in application behavior even though the application code for the Version 1.1 beans has not been changed.

The following information generally applies to any enterprise bean that currently complies with Version 1.1 of the EJB specification. For more information about migrating code for beans produced with the Rational Application Developer tool, see the documentation for that product. For more information about migrating code in general, see ″Resources for learning.″

1. In beans with container-managed persistence (CMP) version 1.x, replace each CMP field with abstract get and set methods. In doing so, you must make each bean class abstract.
2. In beans with CMP version 1.x, change all occurrences of `this.`*`field`* `=` *`value`* to `set`*`Field`*`(`*`value`*`)`.
3. In each CMP bean, create abstract get and set methods for the primary key.
4. In beans with CMP version 1.x, create an EJB Query Language statement for each finder method.

   Note: EJB Query Language has the following limitations in Application Developer Version 5:
   - EJB Query Language queries involving beans with keys made up of relationships to other beans appear as invalid and cause errors at deployment time.
   - The IBM EJB Query Language support extends the EJB 2.1 specification in various ways, including relaxing some restrictions, adding support for more DB2 functions, and so on. If portability across various vendor databases or EJB deployment tools is a concern, then care should be taken to write all EJB Query Language queries strictly according to instructions described in Chapter 11 of the EJB 2.1 specification.
5. In finder methods for beans with CMP version 1.x, return java.util.Collection instead of java.util.Enumeration.
6. Update handling of non-application exceptions.
   - To report non-application exceptions, throw javax.ejb.EJBException instead of java.rmi.RemoteException.
   - Modify rollback behavior as needed: In EJB versions 1.1 and 2.1, all non-application exceptions thrown by the bean instance result in the rollback of the transaction in which the instance is running; the instance is discarded. In EJB 1.0, the container does not roll back the transaction or discard the instance if it throws java.rmi.RemoteException.
7. Update rollback behavior as the result of application exceptions.
   - In EJB versions 1.1 and 2.1, an application exception does not cause the EJB container to automatically roll back a transaction.
   - In EJB Version 1.1, the container performs the rollback only if the instance has called setRollbackOnly() on its EJBContext object.
   - In EJB Version 1.0, the container is required to roll back a transaction when an application exception is passed through a transaction boundary started by the container.
8. Update any CMP setting of application-specific default values to be inside ejbCreate (not using global variables, since EJB 1.1 containers set all fields to generic default values before calling ejbCreate, which overwrites any previous application-specific defaults). This approach also works for EJB 1.0 CMPs.

   Note: In Application Developer Version 5, there is a J2EE Migration wizard to migrate the EJB beans within an EJB 2.1 project from 1.x into 2.1 (you cannot just migrate individually selected beans).

The wizard performs migration steps #1 to #2 above. It also migrates EJB 1.1 (proprietary) relationships into EJB 2.1 (standard) relationships, and maintains EJB inheritance.

Adjusting exception handling for EJB wrappered applications migrating from version 5 to version 6

Because of a change in the Java APIs for XML based Remote Procedure Call (JAX-RPC) specification, EJB applications that could be wrappered in WebSphere Application Server version 5.1 cannot be wrappered in version 6 unless you modify the code to the exception handling of the base EJB application.

Essentially, the JAX-RPC version 1.1 specification states:

```
a service specific exception declared in a remote method signature must be a
checked exception. It must extend java.lang.Exception either directly or indirectly
but it must not be a RuntimeException.
```

So it is no longer possible to directly use java.lang.Exception or java.lang.Throwable types. You must modify your applications using service specific exceptions to comply with the specification.

1. Modify your applications that use service specific exceptions. For example, say that your existing EJB uses a service specific exception called UserException. Inside of UserException is a field called *ex* that is type java.lang.Exception. To successfully wrapper your application with Web services in WebSphere Application Server version 6, you must change the UserException class . In this example, you could modify UserException to make the type of *ex* to be *java.lang.String* instead of java.lang.Exception.

```
new UserException class:

package irwwbase;

/**
 * Insert the type's description here.
 * Creation date: (9/25/00 2:25:18 PM)
 * @author: Administrator
 */


public class UserException extends java.lang.Exception {

       private java.lang.String _infostring = null;
       private java.lang.String ex;
/**
 * UserException constructor comment.
 */

public UserException() {
       super();
}
/**
 * UserException constructor comment.
 */
public UserException (String infostring)
{
       _infostring = infostring;
} // ctor
/**
 * Insert the method's description here.
 * Creation date: (11/29/2001 9:25:50 AM)
 * @param msg java.lang.String
 * @param ex java.lang.Exception
 */
public UserException(String msg,String t) {
       super(msg);
       this.setEx(t);

       }
       /**
        * @return
```

```
         */
        public java.lang.String get_infostring() {
               return _infostring;
        }

        /**
         * @return
         */
        public java.lang.String getEx() {
               return ex;
        }

        /**
         * @param string
         */
        public void set_infostring(java.lang.String string) {
               _infostring = string;
        }

        /**
         * @param Exception
         */
        public void setEx(java.lang.String exception) {
               ex = exception;
        }

        public void printStackTrace(java.io.PrintWriter s) {
          System.out.println("the exception is :"+ex);
          }

}
```

2. Modify all of the exception handling in the enterprise beans that use it. You must ensure that your enterprise beans are coded to accept the new exceptions. In this example, the code might look like this:

```
new EJB exception handling:

try {
      if (isDistributed()) itemCMPEntity = itemCMPEntityHome.findByPrimaryKey(ckey);
      else itemCMPEntityLocal = itemCMPEntityLocalHome.findByPrimaryKey(ckey);
  } catch (Exception ex) {
       System.out.println("%%%%% ERROR: getItemInstance - CMPjdbc " + _className);
       ex.printStackTrace();
       throw new UserException("error on itemCMPEntityHome.findByPrimaryKey(ckey)",ex.getMessage());
   }
```

Container interoperability

Container interoperability describes the ability of WebSphere Application Server clients and servers at different versions to successfully negotiate differences in native Enterprise JavaBeans (EJB) finder methods support and Java 2 Platform, Enterprise Edition (J2EE) compliance.

Interoperability of the handle formats in WebSphere Application Server, Version 5 and Version 5.0.1

Applications that attempt to persist handles to enterprise beans and **EJBHome** needed to subclass ObjectInputStream in WebSphere Application Server, Version 5. This action was required so that the subclass ObjectInputStream could utilize the context class loader to resolve the classes for enterprise beans and EJBHome stubs.

In addition, handles created and persisted in WebSphere Application Server, Version 5 only work with objects that have an unchanged remote interface. If the remote interface is changed, the handle is no longer valid because the stub is serialized inside the handle and its serial Version UID changes if the remote interface changes.

This release introduces a new handle persistence mechanism that avoids the implementation drawbacks of the previous version. However, if handles are used for this WebSphere Application Server deployment, you should consider the following issues when applying this update, future WebSphere Application Server Fix Packs and EJB Container cumulative fixes for WebSphere Application Server, Version 5.

If a WebSphere Application Server, Version 5 persisted handle or home handle is encountered by a WebSphere Application Server, Version 5.0.1 system, it can be read and utilized. In addition, it will be converted to WebSphere Application Server, Version 5.0.1 format if it is re-persisted. The WebSphere Application Server, Version 5.0.1 format cannot be read by a WebSphere Application Server, Version 5 system unless PQ72184 is applied.

Problems arise when handles are persisted and shared across systems that are not at the WebSphere Application Server, Version 5.0.1 level or later. However, a Version 5 system can receive a handle from Version 5.0.1 remotely through a call to get a handle on an enterprise bean or a getHomeHandle on an **EJBHome**. The remote call will succeed, however, any attempt to persist it on the Version 5 system will have the same limitations regarding the use of ObjectInputStream and changes in remote interface invalidating the persisted handle.

When your application stores handles persistently and shares this persistence with multiple clients or application servers, apply WebSphere Application Server, Version 5.0.1 or PQ72184 to both the client and server systems at the same time. Failure to do so can result in the inability of these systems to read the handle data stored by upgraded systems. Also, handles stored by the WebSphere Application Server, Version 5 can force the applications of the updated system to still subclass `ObjectInputStream`. Applications using the WebSphere Application Server Enterprise, Version 5 scheduler and process choreographer, are affected by these changes. These users should update their Version 5 systems at the same time with either Version 5.0.1 or PQ72184.

If the applications store handles in the session context, or locally in a file on the same system, that is not shared by other applications, on different systems, they might be able to update their systems individually, rather than all at once. If Client Container and thin client applications do not share persisted handle data, they can be updated as needed as well. However, handles created and persisted in WebSphere Application Server, Version 5, Version 4.0.3 and later (with the property flag set), or Version 3.5.7 and later (with the property flag set) are not usable if either the home or the remote interface changes.

If any WebSphere Application Server, Version 3.5.7 or Version 4.0.3 and later enables the system property com.ibm.websphere.container.portable to **true**, any handles to objects on that server have the same interoperability limitations. In addition, if any WebSphere Application Server, Version 3.5.7 and later or Version 4.0.3 applications store a handle obtained from a WebSphere Application Server, Version 5 or Version 5.0.1, the same restrictions apply, regarding the need to subclass ObjectInputStream and the usability of handles after a change to the remote interface is made.

Replication of the Http Session and Handles

This note applies to you if you place Handles to Homes or Enterprise JavaBeans, or EJB or EJBHome references in the Http Session in your application and you use Http Session Replication. If you intend to replicate a mixed environment of Version 5.0.0 and Version 5.0.1 or 5.0.2 machines you should first apply the latest Version 5.0.0 container cumulative e-fix to the Version 5.0.0 machines before allowing the Version 5.0.1 or 5.0.2 server into the typology. The reason for this is that Version 5.0.0 servers are not able to understand the persisted Handle format used on the Version 5.0.1 and 5.0.2 server. This is similar to the case of Version 5.0.0 and Version 5.0.1 or 5.0.2 systems trying to use a shared database, mentioned above. But in this case, it is the Http Session object and not the database providing the persistence.

Top Down Deployment Mapping

The size of the Handle objects has grown due to the fix put in to allow serialization and deserialization to occur without the previous requirements of subclassing the ObjectInputStream and so on. Top down deployment of an object that contains EJB and EJBHome references create a database table ddl that has a field of 1000 bytes of VARCHAR for BITDATA which will contain the Handle. It might be that your object's Handle does not fit in the 1000 byte default field, and you might need to adjust this to a higher value. You might try increments of 250 bytes, that is, 1250, 1500, and so on.

Web services

Web Services-Interoperability Basic Profile

The *Web Services-Interoperability (WS-I) Basic Profile* is a set of non-proprietary Web services specifications that promote interoperability.

WebSphere Application Server conforms to the WS-I Basic Profile 1.1.

The WS-I Basic Profile is governed by a consortium of industry-leading corporations, including IBM, under direction of the WS-I Organization. The profile consists of a set of principles that relate to bringing about open standards for Web services technology. All organizations that are interested in promoting interoperability among Web services are encouraged to become members of the Web Services Interoperability Organization.

Several technology components are used in the composition and implementation of Web services, including messaging, description, discovery, and security. Each of these components are supported by specifications and standards, including SOAP 1.1, Extensible Markup Language (XML) 1.0, HTTP 1.1, Web Services Description Language (WSDL) 1.1, and Universal Description, Discovery and Integration (UDDI). The WS-I Basic Profile specifies how these technology components are used together to achieve interoperability, and mandates specific use of each of the technologies when appropriate. You can read more about the WS-I Basic Profile at the WS-I Organization Web site.

Each of the technology components have requirements that you can read about in more detail at the WS-I Organization Web site. For example, support for Universal Transformation Format (UTF)-16 encoding is required by WS-I Basic Profile. UTF-16 is a kind of Unicode encoding scheme using 16-bit values to store Universal Character Set (UCS) characters. UTF-8 is the most common encoding that is used on the Internet; UTF-16 encoding is typically used for Java and Windows product applications; and UTF-32 is used by various Linux and Unix systems. Unlike UTF-8, UTF-16 has issues with big-endian and little-endian, and often involves Byte Order Mark (BOM) to indicate the endian. BOM is mandatory for UTF-16 encoding and it can be used in UTF-8.

See how to modify your encoding from UTF-8 to UTF-16 if you need to change from UTF-8 to UTF-16.

The following table summarizes some of the properties of each UTF:

Bytes	Encoding form
EF BB BF	UTF-8
FF FE	UTF-16, little-endian
FE FF	UTF-16, big-endian
00 00 FE FF	UTF-32, big-endian
FF FE 00 00	UTF-32, little-endian

BOM is written prior to the XML text, and it indicates to the parser how the XML is encoded. The XML declaration contains the encoding, for example: <?xml version=*xxx* encoding="utf-*xxx*"?>. BOM is used with the encoding to determine how to interpret the XML. Here is an example of a SOAP message and how BOM and UTF encoding are used:

```
POST http://www.whitemesa.net/soap12/add-test-rpc HTTP/1.1
Content-Type: application/soap+xml; charset=utf-16; action=""
SOAPAction:
Host: localhost: 8080
Content-Length: 562

0xFF0xFE<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/12/soap-envelope"
 xmlns:soapenc="http://www.w3.org/2002/12/soap-encoding
 xmlns:tns="http://whitemesa.net/wsdl/soap12-test"
 xmlns:types="http://whitemesa.net/wsdl/soap12-test/encodedTypes"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <q1:echoString xmlns:q1="http://soapinterop.org/">
   <inputString soap:encodingStyle="http://example.org/unknownEncoding"
   xsi:type="xsd:string">
    Hello SOAP 1.2
   </inputString>
  </q1:echoString>
 </soap:Body>
</soap:Envelope>
```

In the example code, 0xFF0xFE represents the byte codes, while the *<?xml>* declaration is the textual representation.

How to change encoding in a SOAP message to support WSI-Basic Profile

Support for Universal Transformation Format (UTF)-16 encoding is required by the WS-I Basic Profile 1.0. WebSphere Application Server conforms to the WS-I Basic Profile 1.1. UTF-16 is a kind of unicode encoding scheme using 16-bit values to store Universal Character Set (UCS) characters. UTF-8 is the most common encoding that is used on the Internet and UTF-16 encoding is typically used for Java and Windows product applications. This task teaches you how to change the encoding in a SOAP message from UTF-8 to UTF-16.

To learn more about the requirements of the Web Services-Interoperability Basic Profile (WS-I), including UTF-16, see Web Services-Interoperability Basic Profile.

Support for UTF-16 encoding is required by WS-I Basic Profile; therefore, you need to know how to modify your character encoding from UTF-8 to UTF-16 in your SOAP message.

You can change the character encoding in one of two ways:

- Use a property on the Stub for users to set.

  This choice applies to the client only.

  For a client, the encoding is specified in the SOAP request. The SOAP engine serializes the request and sends it to the Web service engine. The Web service engine receives the request and deserializes the message to Java objects, which are returned to you in a response.

  When the Web service engine on the server receives the serialized request, a raw message in the form of an input stream, is passed to the parser, which understands Byte Order Mark (BOM). BOM is mandatory for UTF-16 encoding and it can be used in UTF-8. The message is deserialized to a Java objects and a service invocation is made. For two-way invocation, the engine needs to serialize the message using a specific encoding and send it back to the caller. The following example shows you how to use a property on the Stub to change the character set:

  ```
  javax.xml.rpc.Stub stub=service.getPort("MyPortType");
  ((javax.xml.rpc.Stub)stub).setProperty(com.ibm.wsspi.webservices.Constants.MESSAGE_CHARACTER_SET_ENCODING,"UTF-16");
  stub.invokeMethod();
  ```

In this code example, **com.ibm.wsspi.webservices.Constants.MESSAGE_CHARACTER_SET_ENCODING** = *"com.ibm.wsspi.webservices.xmlcharset";*

- Use a handler to change the character set through SOAP with Attachments API for Java (SAAJ).

  If you are using a handler, the SOAP message is transformed to a SAAJ format from other possible forms, such as an input stream. In such cases as a handleRequest method on the client side and a handleResponse method on the server side, the Web services engine transforms from a SAAJ format back to the stream with appropriate character encoding. This transformation or change is called a *roundtrip transformation*. The following is an example of how you would use a handler to specify the character encoding through SAAJ:

```
handleResponse(MessageContext mc) {
 SOAPMessageContext smc = (SOAPMessageContext) context;
 javax.xml.soap.SOAPMessage msg = smc.getMessage();
 msg.setProperty (javax.xml.soap.SOAPMessage.CHARACTER_SET_ENCODING, "UTF-16");
 }
}
```

You have modified the character encoding from UTF-8 to UTF-16 in the Web service SOAP message.

Migrating Web services

This topic explains information that you need to migrate Web services.

If you have used the Apache SOAP support to develop Web services client applications in WebSphere Application Server Versions 4, 5, or 5.1, you might need to migrate your applications or the security files for your applications. The following table summarizes the Web services specifications supported by the WebSphere products.

Table 1.

WebSphere Application Server Version	Web services specifications supported
4.0	SOAP 2.2
5.0 and 5.0.1	SOAP 2.3
5.0.2 or later	Java 2 Platform, Enterprise Edition (J2EE), also known as (JSR 109)
6.0	J2EE (JSR 109)

The SOAP 2.2 and SOAP 2.3-based implementations that were available in WebSphere Application Server Version 4.0.x, 5.0 and 5.0.1 have been deprecated. It is recommended that applications that are using these SOAP implementations migrate to Web Services for J2EE (JSR 109) support that is provided in current WebSphere Application Server versions.

For more information on migrating your Web services, see Migrating Apache SOAP Web services to Web Services to J2EE standards and Web services migration best practices.

It is recommended that new Web services be developed using the Web services for J2EE specification. For more information, see the article on implementing Web services applications in the information center.

Security cannot be directly migrated from SOAP 2.3 to the J2EE standards. After you have migrated your Web services to the J2EE standards, see the article on securing Web services for Version 6 applications based on WS-Security in the information center.

Migrating Apache SOAP Web services to Web Services for J2EE standards

This topic explains how to migrate Web services that were developed using Apache SOAP to Web services that are developed based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification.

If you have used Web services based on Apache SOAP and now want to develop and implement Web services based on the Web Services for J2EE specification, you need to migrate client applications developed with all versions of 4.0, and versions of 5.0 prior to 5.0.2.

To migrate these client applications according to the Web Services for J2EE standards:

1. Plan your migration strategy. You can port an Apache SOAP client to a Java API for XML-based RPC (JAX-RPC) Web services client in one of two ways:
   - If you have, or can create, a Web Services Description Language (WSDL) document for the service, consider using the **WSDL2Java** command tool to generate bindings for the Web service. It is more work to adapt an Apache SOAP client to use the generated JAX-RPC bindings, but the resulting client code is more robust and easier to maintain.

     To follow this path, see the article on developing a Web services client based on Web Services for J2EE in the information center.
   - If you do not have a WSDL document for the service, do not expect the service to change, and you want to port the Apache SOAP client with minimal work, you can convert the code to use the JAX-RPC dynamic invocation interface (DII), which is similar to the Apache SOAP APIs. The DII APIs do not use WSDL or generated bindings.

   Because JAX-RPC does not specify a framework for user-written serializers, the JAX-RPC does not support the use of custom serializers. If your application cannot conform to the default mapping between Java, WSDL, and XML technology supported by WebSphere Application Server, do not attempt to migrate the application. The remainder of this topic assumes that you decided to use the JAX-RPC dynamic invocation interface (DII) APIs.
2. Review the GetQuote Sample. A Web services migration Sample is available in the Samples Gallery. This Sample is located in the `GetQuote.java` file, originally written for Apache SOAP users, and includes an explanation about the changes needed to migrate to the JAX-RPC DII interfaces.
3. Convert the client application from Apache SOAP to JAX-RPC DII The Apache SOAP API and JAX-RPC DII API structures are similar. You can instantiate and configure a call object, set up the parameters, invoke the operation, and process the result in both. You can create a generic instance of a Service object with the following command:

   ```
   javax.xml.rpc.Service service = ServiceFactory.newInstance().createService(new QName(""));
   ```

   in JAX-RPC.

   a. Create the Call object. An instance of the Call object is created with the following code:

      ```
      org.apache.soap.rpc.Call call = new org.apache.soap.rpc.Call ()
      ```

      in Apache SOAP.

      An instance of the Call object is created by

      ```
      java.xml.rpc.Call call = service.createCall();
      ```

      in JAX-RPC.

   b. Set the endpoint Uniform Resource Identifiers (URI). The target URI for the operation is passed as a parameter to

      ```
      call.invoke:  call.invoke("http://...", "");
      ```

      in Apache SOAP.

      The setTargetEndpointAddress method is used as a parameter to

      ```
      call.setTargetEndpointAddress("http://...");
      ```

      in JAX-RPC.

      Apache SOAP has a setTargetObjectURI method on the Call object that contains routing information for the request. JAX-RPC has no equivalent method. The information in the targetObjectURI is included in the targetEndpoint URI for JAX-RPC.

c. Set the operation name. The operation name is configured on the Call object by

```
call.setMethodName("opName");
```

in Apache SOAP.

The setOperationName method, which accepts a `QName` instead of a `String` parameter, is used in JAX-RPC as illustrated in the following example:

```
call.setOperationName(new javax.xml.namespace.Qname("namespace", "opName"));
```

d. Set the encoding style. The encoding style is configured on the Call object by

```
call.setEncodingStyleURI(org.apache.soap.Constants.NS_URI_SOAP_ENC);
```

in Apache SOAP.

The encoding style is set by a property of the Call object

```
call.setProperty(javax.xml.rpc.Call.ENCODINGSTYLE_URI_PROPERTY, "http://schemas.
xmlsoap.org/soap/encoding/");
```

in JAX-RPC.

e. Declare the parameters and set the parameter values. Apache SOAP parameter types and values are described by parameter instances, which are collected into a vector and set on the Call object before the call, for example:

```
Vector params = new Vector ();
params.addElement (new org.apache.soap.rpc.Parameter(name, type, value, encodingURI));
// repeat for additional parameters...
call.setParams (params);
```

For JAX-RPC, the Call object is configured with parameter names and types without providing their values, for example:

```
call.addParameter(name, xmlType, mode);
// repeat for additional parameters
call.setReturnType(type);
```

Where

- *name* (type `java.lang.String`) is the name of the parameter
- *xmlType* (type `javax.xml.namespace.QName`) is the XML type of the parameter
- *mode* (type `javax.xml.rpc.ParameterMode`) the mode of the parameter, for example, IN, OUT, or INOUT

f. Make the call. The operation is invoked on the Call object by

```
org.apache.soap.Response resp = call.invoke(endpointURI, "");
```

in Apache SOAP.

The parameter values are collected into an array and passed to `call.invoke` as illustrated in the following example:

```
Object resp = call.invoke(new Object[] {parm1, parm2,...});
```

in JAX-RPC.

g. Check for faults. You can check for a SOAP fault on the invocation by checking the response:

```
if resp.generatedFault then {
org.apache.soap.Fault f = resp.getFault;
f.getFaultCode();
f.getFaultString();
}
```

in Apache SOAP.

A `java.rmi.RemoteException` error is displayed in JAX-RPC if a SOAP fault occurs on the invocation.

```
try {
... call.invoke(...)
} catch (java.rmi.RemoteException) ...
```

h. Retrieve the result. In Apache SOAP, if the invocation is successful and returns a result, it can be retrieved from the Response object:

```
Parameter result = resp.getReturnValue(); return result.getValue();
```

In JAX-RPC, the result of invoke is the returned object when no exception is displayed:

```
Object result = call.invoke(...);
 ...
return result;
```

You have migrated Apache SOAP Web services to J2EE Web services.

Develop a Web services client. See the article in the information center on how to develop a Web services client based on the Web Services for J2EE specification.

Test the Web services-enabled clients to make sure that the migration process is successful and you can implement the Web services in a J2EE environment.

Web services migration best practices

This topic presents best practices for migrating Web services applications.

Migrating a Version 5 Java API for XML-based remote procedure call (JAX-RPC) client that uses SOAP over Java Message Service (JMS) to invoke a Web service

A JAX-RPC client that is executed on WebSphere Application Server Version 5, can use SOAP over JMS to invoke a Web service that is executed on a Version 5 application server.

A user ID and password are not required on the target WebSphere MQ queue. After the application server is migrated to Version 6.x, and uses the Version 6.x default messaging feature, client requests can fail because basic authentication is enabled. The following error message displays when this migration problem occurs:

```
SibMessage W [:] CWSIT0009W: A client request failed in the application server with
endpoint <endpoint name> in bus <bus_name> with reason: CWSIT0016E: The user
ID null failed authentication in bus <bus_name>.
```

When the application server is migrated to Version 6.x, and the default messaging provider (service integration technologies) is used, and administrative and application security is enabled for the server or the cell, the service integration bus queue destination inherits the security characteristics of the server or the cell by default. If the server or the cell has basic authentication enabled, the client request fails.

The following options are available to solve this problem. The solutions are listed by the level of security that they impose:

- Disable administrative and application security on the main security panel within the administrative console. To disable administrative and application security, click **Security** > **Secure administration, applications and infrastructure**. Deselect the **Enable administrative security** and **Enable application security** options.
- Modify the settings for the service integration bus that hosts the queue destination so that the bus security is disabled and the bus does not inherit security characteristics from the server or the cell. This option is equivalent to the level of security that you can configure in Version 5.
- Configure the basic authentication on each client that uses the service.

Migrating Apache SOAP Web services

See Migrating Apache SOAP Web services to Web Services for J2EE standards to learn how to migrate Apache SOAP Web services. This topic explains how to migrate Web services that were developed using Apache SOAP to Web services that are developed based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification.

Migrating Web services assembled with early versions of the Application Server Toolkit or Assembly Toolkit

If you are migrating your Web service or Web service components from earlier versions of the Application Server Toolkit or Assembly Toolkit, refer to the following hints and tips to improve your success:

- Secure Web services are not migrated by the J2EE Migration Wizard when Web services are migrated from J2EE 1.3 to J2EE 1.4.
- The migration of secure Web services requires manual steps.
- After the J2EE migration, the secure binding and extension files must be migrated manually to J2EE 1.4 as follows:
  1. Double click on the `webservices.xml` file to open the Web Services editor.
  2. Select the **Binding Configurations** tab to edit the binding file.
  3. Add all the necessary binding configurations under the new sections **Request Consumer Binding Configuration Details** and **Response Generator Binding Configuration Details**.
  4. Select the **Extension** tab to edit the extension file.
  5. Add all the necessary extension configurations under the new sections **Request Consumer Service Configuration Details** and **Response Generator Service Configuration Details**.
  6. Save and exit the editor.

.

Migrating the UDDI registry

With most scenarios, migration of existing UDDI registries happens automatically when you migrate to the current level of WebSphere Application Server. However, if your existing UDDI registry uses a network Cloudscape database or a DB2 UDDI Version 2 database, there are some manual steps that you must take.

Migrate your installation of WebSphere Application Server; ensure that you select the option to migrate applications, so that the UDDI registry application will be migrated.

If your existing UDDI registry uses an Oracle, embedded Cloudscape or DB2 UDDI Version 3 database, you do not need to perform any manual migration; migration happens automatically when you migrate WebSphere Application Server and start the UDDI node for the first time after migration.

If your existing UDDI registry uses a network Cloudscape database or a DB2 UDDI Version 2 database, you must complete some manual steps to migrate the registry.

- If your UDDI registry uses a DB2 UDDI Version 2 database, follow the steps in “Migrating to Version 3 of the UDDI registry” on page 39 and sub-topics.
- If your UDDI registry uses a network Cloudscape database, complete the following steps.
  1. If you have a cluster that contains servers at different levels of WebSphere Application Server, ensure that any UDDI registries are running on servers that are at WebSphere Application Server Version 6.1. For example, if you have a cluster that spans two nodes, you can upgrade one node to WebSphere Application Server Version 6.1 while the other node remains at a previous level, provided that any servers that are running a UDDI registry are at Version 6.1.
  2. Initialize the relevant UDDI node. The initialize process will perform some of the UDDI registry migration.

3. Enter the following commands as the database administrator, from *app_server_root*/cloudscape/lib.

```
java -cp db2j.jar;db2jtools.jar com.ibm.db2j.tools.ij

connect 'jdbc:db2j:uddi_cloudscape_database_path';

run 'app_server_root/UDDIReg/databaseScripts/uddi30crt_drop_triggers_cloudscape.sql';

quit;

cd app_server_root/derby/migration

java -cp db2j.jar;db2jmigration.jar;../lib/derby.jar com.ibm.db2j.tools.MigrateFrom51
  jdbc:db2j:uddi_cloudscape_database_path
```

   where

   – *uddi_cloudscape_database_path* is the absolute path of the existing Cloudscape database, for example *app_server_root*/profiles/*profile_name*/databases/com.ibm.uddi/UDDI30
   – *app_server_root* is the root directory for the installation of WebSphere Application Server

The UDDI database and data source are migrated, and the UDDI node is activated.

Note: When you migrate WebSphere Application Server, the post-upgrade log for the profile indicates that the migration of the UDDI database is partially complete, and is missing the steps for triggers, aliases, and stored statements. If you initially enabled the debug function, the debug log for the database indicates that there was a failure creating triggers. Ignore these messages; the UDDI node completes the migration of the database when the UDDI node starts. For more information about these log files, see “Verifying the Cloudscape v10.1.x automatic migration” on page 45. Also refer to this topic if other errors appear in the logs.

If the migration of the UDDI database completes successfully, the following message appears in the server log:

```
CWUDQ0003I: UDDI registry migration has completed
```

If the following error appears, an unexpected error occurred during migration. The UDDI registry node is not activated. Check the error logs for the problem and, if it cannot be fixed, see the IBM software support Web site at http://www.ibm.com/software/support.

```
CWUDQ004W: UDDI registry not started due to migration errors
```

Migrating to Version 3 of the UDDI registry

Use this topic to migrate a Version 2 UDDI registry that uses a DB2 database, running in WebSphere Application Server Version 5 or later, to a Version 3 UDDI registry running in WebSphere Application Server Version 6.x.

The following constraints apply to this procedure:

- Your existing registry must use a DB2 database.
- Your existing registry must run in WebSphere Application Server Version 5 or later. If you are migrating from UDDI registry Version 1.1 or 1.1.1, which ran on WebSphere Application Server Version 4, first migrate to UDDI Version 2 running on WebSphere Application Server Version 5 (as described in the WebSphere Application Server Version 5 Information Center), then complete the steps described in this topic.

1. Stop the UDDI registry application that is running in your Version 5.x application server. This prevents further UDDI requests being directed to the UDDI registry and ensures that no new data is published during the migration process.

2. Record information about the uddi.properties values being used. This file is located in the *DeploymentManager_install_dir*/config/cells/*cell_name*/nodes/*node_name*/servers/*server_name* directory on your WebSphere Application Server Version 5.x system (or in the properties subdirectory if you are migrating a standalone application server).
3. Migrate the server from WebSphere Application Server Version 5.x to Version 6.x. This results in a new directory tree for the migrated Version 6.x application server.
4. Start the new, migrated Version 6.x application server.
5. Create a new data source for the Version 2 UDDI database. This data source is known as the *UDDI migration data source*. The JNDI name must be `datasources/uddimigration`. To complete this step see Setting up a UDDI migration datasource.
6. Set up the UDDI Version 3 registry and migrate the Version 2 data.

   Follow the instructions in the topic in the information center on setting up a customized UDDI node, including the subtopic that relates to node initialization. The topic describes how to perform the following actions:
   - Create the Version 3 DB2 database
   - Create the J2C authentication data entry
   - Create the JDBC provider and data source
   - Deploy the UDDI registry application
   - Start the server
   - Configure and initialize the node. The UDDI registry node initialization detects the UDDI migration data source, and migrates the Version 2 data as part of the UDDI node initialization processing. This data migration can take some time, depending on the amount of data in your UDDI registry.

The UDDI registry is migrated. If the UDDI node remains in a state of initialization pending, migration pending or value set creation pending, check the server log for errors. If the following message appears, an unexpected error occurred during migration. Check the error logs for the problem and, if it cannot be fixed, see the IBM software support Web site at http://www.ibm.com/software/support.

```
CWUDQ004W: UDDI registry not started due to migration errors
```

After the problem is fixed, you can complete the migration by clicking **Initialize** again.

Verify that the migration process completed successfully by checking for the following message in the server log:

```
CWUDQ0003I: UDDI registry migration has completed
```

After migration is complete you can remove the UDDI migration data source, and the registry is available for use.

Setting up a UDDI migration datasource

Use this topic to set up a UDDI migration datasource, to be used to reference a Version 2 UDDI registry database.

Migration is only supported from DB2, so these instructions describe how to set up a DB2 datasource.

1. If a suitable JDBC Provider for DB2 does not already exist, then create one, selecting the options DB2 Universal JDBC Driver Provider and Connection Pool datasource.

   For details on how to create a JDBC provider, see the article in the information center on creating and configuring a JDBC provider using the administrative console.
2. Create a datasource for the Version 2 UDDI registry by following these steps:
   a. Expand **Resources** and **JDBC Providers**.
   b. Click **Resources** → **JDBC** → **JDBC Providers**.

c. Select the desired 'scope' of the JDBC provider you selected or created earlier. For example, select:

```
Server: yourservername
```

to show the JDBC providers at the server level.

d. Select the JDBC provider created earlier.

e. Under **Additional Properties**, select **Data sources** (*not* the **Data sources (WebSphere Application Server V4)** option).

f. Click **New** to create a new datasource.

g. In the **Create a data source** wizard, enter the following data:

Name a suitable name, such as UDDI Datasource

JNDI name
set to **datasources/uddimigration** - this value is compulsory, and must be as shown.

Component-managed authentication alias
select the alias for the DB2 userid used to access UDDI Version 2 data, for example MyNode/UDDIAlias

h. Click **Next**.

i. On the database specific properties page of the wizard, enter the following data:

Database name
`UDDI20`, or the name given to your Version 2 UDDI DB2 database.

Use this Data Source in container-managed persistence (CMP)
ensure the check box is cleared.

j. Click **Next**, then check the summary and click **Finish**.

k. Click the data source to display its properties, and add the following information:

Description
a suitable description

Category
set to **uddi**

Data store helper class name
filled in for you as: com.ibm.websphere.rsadapter.DB2DataStoreHelper

Mapping-configuration alias
set to DefaultPrincipalMapping

3. Click **Apply** and save the changes to the master configuration.

4. Test the connection to your UDDI database by selecting the check box next to the datasource and clicking **Test connection**. You will see a message similar to ″Test Connection for datasource UDDI Datasource on server server1 at node MyNode was successful″. If you do not see this message investigate the problem with the help of the error message.

Continue with the migration as detailed in Migrating to Version 3 of the UDDI registry.

Data access resources

Migrating applications to use data sources of the current J2EE Connector Architecture (JCA)

Migrate your applications that use Version 4 data sources, or data sources (WebSphere Application Server V4), to use data sources that support more advanced connection management features, such as connection sharing.

To use the connection management infrastructure in WebSphere Application Server Version 6.x, you must package your application as a J2EE 1.3 (or later) application. This process involves repackaging your Web modules to the 2.3 specification and your EJB modules to the 2.1 specification before installing them onto WebSphere Application Server.

In WebSphere Application Server Version 6.x, data sources are intended for use within J2EE applications and designed to operate within the EJB and Web containers.

Converting a 2.2 Web module to a 2.3 Web module

Use the following steps to migrate each of your Web modules.

1. Open an assembly tool such as the Application Server Toolkit (AST) or Rational Web Developer.
2. Create a new Web module by selecting **File > New > Web Module**.
3. Add any required class files to the new module.
   a. Expand the **Files** portion of the tree.
   b. Right-click **Class Files** and select **Add Files**.
   c. In the Add Files window, click **Browse**.
   d. Navigate to your WebSphere Application Server 4.0 EAR file and click **Select**.
   e. In the upper left pane of the Add Files window, navigate to your WAR file and expand the `WEB-INF` and `classes` directories.
   f. Select each of the directories and files in the classes directory and click **Add**.
   g. After you add all of the required class files, click **OK**.
4. Add any required JAR files to the new module.
   a. Expand the **Files** portion of the tree.
   b. Right-click **Jar Files** and select **Add Files**.
   c. Navigate to your WebSphere 4.0 EAR file and click **Select**.
   d. In the upper left pane of the Add Files window, navigate to your WAR file and expand the `WEB-INF` and `lib` directories.
   e. Select each JAR file and click **Add**.
   f. After you add all of the required JAR files, click**OK**.
5. Add any required resource files, such as HTML files, JSP files, GIFs, and so on, to the new module.
   a. Expand the **Files** portion of the tree.
   b. Right-click **Resource Files** and select **Add Files**.
   c. Navigate to your WebSphere Application Server 4.0 EAR file and click **Select**.
   d. In the upper left pane of the Add Files window, navigate to your WAR file.
   e. Select each of the directories and files in the WAR file, excluding `META-INF` and `WEB-INF`, and click **Add**.
   f. After you add all of the required resource files, click**OK**.
6. Import your Web components.
   a. Right-click **Web Components** and select **Import**.
   b. In the Import Components window click **Browse**.
   c. Navigate to your WebSphere Application Server 4.0 EAR file and click **Open**.
   d. In the left top pane of the **Import Components** window, highlight the WAR file that you are migrating.
   e. Highlight each of the components that display in the right top pane and click **Add**.
   f. When all of your Web components display in the Selected Components pane of the window, click **OK**.
   g. Verify that your Web components are correctly imported under the Web Components section of your new Web module.

7. Add servlet mappings for each of your Web components.
   a. Right-click **Servlet Mappings** and select **New**.
   b. Identify a URL pattern for the Web component.
   c. Select the web component from the Servlet drop-down box.
   d. Click **OK**.
8. Add any necessary resource references by following the instructions in the Creating a resource reference article in the information center.
9. Add any other Web module properties that are required. Click **Help** for a description of the settings.
10. **Save** the Web module.

Converting a 1.1 EJB module to a 2.1 EJB module (or later)

Use the following steps to migrate each of your EJB modules.

1. Open an assembly tool.
2. Create a new EJB Module by selecting **File > New > EJB Module**.
3. Add any required class files to the new module.
   a. Right-click **Files object** and select **Add Files**.
   b. In the Add Files window click **Browse**.
   c. Navigate to your WebSphere Application Server 4.0 EAR file and click **Select**.
   d. In the upper left pane of the Add Files window, navigate to your enterprise bean JAR file.
   e. Select each of the directories and class files and click **Add**.
   f. After you add all of the required class files, click **OK**.
4. Create your session beans and entity beans. To find help on this subject, see Migrating enterprise bean code to the supported specification, the documentation for Rational Application Developer, or the documentation for WebSphere Studio Application Developer Integration Edition.
5. Add any necessary resource references by following the instructions in the Creating a resource reference article in the information center.
6. Add any other EJB module properties that are required. Click **Help** for a description of the settings.
7. **Save** the EJB module.
8. Generate the deployed code for the EJB module by clicking **File > Generate Code for Deployment**.
9. Fill in the appropriate fields and click **Generate Now**.

Add the EJB modules and Web modules to an EAR file

1. Open an assembly tool.
2. Create a new Application by selecting **File > New > Application**.
3. Add each of your EJB modules.
   a. Right-click **EJB Modules** and select **Import**.
   b. Navigate to your converted EJB module and click **Open**.
   c. Click **OK**.
4. Add each of your Web modules.
   a. Right-click **Web Modules** and select **Import**.
   b. Navigate to your converted Web module and click **Open**.
   c. Fill in a **Context root** and click **OK**.
5. Identify any other application properties. Click **Help** for a description of the settings.
6. Save the EAR file.

Installing the Application on WebSphere Application Server

1. Install the application following the instructions in the Installing a new application article in the information center, and bind the resource references to the data source that you created.

2. Perform the necessary administrative task of creating a JDBC provider and a data source object following the instructions in the Creating a JDBC provider and data source article in the information center.

Connection considerations when migrating servlets, JavaServer Pages, or enterprise session beans

Because WebSphere Application Server provides backward compatibility with application modules coded to the J2EE 1.2 specification, you can continue to use Version 4 style data sources when you migrate to Application Server Version 6.x. As long as you configure Version 4 data sources *only* for J2EE 1.2 modules, the behavior of your data access application components does not change.

If you are adopting a later version of the J2EE specification along with your migration to Application Server Version 6.x, however, the behavior of your data access components can change. Specifically, this risk applies to applications that include servlets, JavaServer Page (JSP) files, or enterprise session beans that run inside local transactions over shareable connections. A behavior change in the data access components can adversely affect the use of connections in such applications.

This change affects all applications that contain the following methods:
- `RequestDispatcher.include()`
- `RequestDispatcher.forward()`
- JSP includes (`<jsp:include>`)

Symptoms of the problem include:
- Session hang
- Session timeout
- Running out of connections

Note: You can also experience these symptoms with applications that contain the components and methods described previously if you are upgrading from J2EE 1.2 modules *within* Application Server Version 6.x.

Explanation of the underlying problem

For J2EE 1.2 modules using Version 4 data sources, WebSphere Application Server issues non-sharable connections to JSP files, servlets, and enterprise session beans. All of the other application components are issued shareable connections. However, for J2EE 1.3 and 1.4 modules, Application Server issues shareable connections to *all* logically named resources (resources bound to individual references) unless you specify the connections as unshareable in the individual resource-references. Using shareable connections in this context has the following effects:
- All connections that are received and used outside the scope of a user transaction are *not* returned to the free connection pool until the encapsulating method returns, even when the connection handle issues a `close()` call.
- All connections that are received and used outside the scope of a user transaction are *not* shared with other component instances (that is, other servlets, JSP files, or enterprise beans).

  For example, session bean 1 gets a connection and then calls session bean 2 that also gets a connection. Even if all properties are identical, each session bean receives its own connection.

If you do not anticipate this change in the connection behavior, the way you structure your application code can lead to excessive connection use, particularly in the cases of JSP includes, session beans that run inside local transactions over shareable connections, `RequestDispatcher.include()`routines, `RequestDispatcher.forward()` routines, or calls from these methods to other components.

Examples of the connection behavior change

Servlet A gets a connection, completes the work, commits the connection, and calls **`close()`** on the connection. Next, servlet A calls the `RequestDispatcher.include()` to include servlet B, which performs the

same steps as servlet A. Because the servlet A connection does not return to the free pool until it returns from the current method, two connections are now busy. In this way, more connections might be in use than you intended in your application. If these connections are not accounted for in the **Max Connections** setting on the connection pool, this behavior might cause a lack of connections in the pool, which results in `ConnectionWaitTimeOut` exceptions. If the **connection wait timeout** is not enabled, or if the **connection wait timeout** is set to a large number, these threads might appear to hang because they are waiting for connections that are never returned to the pool. Threads waiting for new connections do not return the ones they are currently using if new connections are not available.

Alternatives to the connection behavior change

To resolve these problems:

1. Use unshared connections.

   If you use an unshared connection and are not in a user transaction, the connection is returned to the free pool when you issue a `close()` call (assuming you commit or roll back the connection).
2. Increase the maximum number of connections.

   To calculate the number of required connections, multiply the number of configured threads by the deepest level of component call nesting (for those calls that use connections). See the Examples section for a description of call nesting.

Verifying the Cloudscape v10.1.x automatic migration

WebSphere Application Server Version 6.1.x requires Cloudscape to run at a minimal version of v10.1.x. (Note that Cloudscape v10.1.x is comprised of the Derby code base.) During the Application Server v6.1.x upgrade, the migration tool automatically upgrades the database instances that are accessed through the embedded framework by some internal components, such as the UDDI registry. The tool also attempts to upgrade Cloudscape instances that your applications access through the embedded framework. You must verify the migration results for these backend databases.

Do not use Cloudscape v10.1.x as a production database. Use it for development and test purposes only.

Learn more: The new version of Cloudscape combines the Derby runtime with additional benefits, such as IBM Quality Assurance (QA) and national language support (NLS). For information about the Cloudscape v10.1.x open source code base, see the Cloudscape section of **ibm**.com: http://www-306.ibm.com/software/data/cloudscape/.

The migration tool attempts to upgrade Cloudscape database instances that are accessed through the embedded framework only. You must manually upgrade Cloudscape instances that transact with application servers on the Network Server framework. (See the “Upgrading Cloudscape manually” on page 47 article.) This requirement eliminates the risk of corrupting third party applications that use the Network Server framework to access the same database instances as WebSphere Application Server.

Cloudscape requires the Network Server framework to transact with more than one Java Virtual Machine (JVM) concurrently. Within WebSphere Application Server, each application server is a JVM; hence you can use a single instance of Cloudscape Network Server framework with a clustered or coexistence implementation of Application Server. For more information on the Network Server framework, consult the IBM Cloudscape information center at http://publib.boulder.ibm.com/infocenter/cscv/v10r1/index.jsp.

For database instances that your applications access through the embedded framework, the automatic migration can succeed completely, fail completely, or succeed with warnings. A migration that produces warning messages does create a Cloudscape v10.1.x database with your data, but does not migrate all of your configured logic and other settings, such as:

- keys
- checks
- views

- triggers
- aliases
- stored statements

To distinguish between a partially and a completely successful migration, you must verify the auto-migration results by checking both the general post-upgrade log and the individual database logs. Performing these tasks gives you vital diagnostic data to troubleshoot the partially migrated databases as well as those that fail auto-migration completely. Ultimately, you migrate these databases through a manual process.

1. Open the post-upgrade log of each new WebSphere Application Server Version 6.1x profile. The path name of the log is WAS_HOME/profiles/*profileName*/logs/WASPostUpgrade.*timestamp*.log.
2. Examine the post-upgrade log for database error messages. These exceptions indicate database migration failures. The following lines are an example of post-upgrade log content, in which the database error code is DSRA7600E. (The migration tool references all database exceptions with the prefix DSRA.)

   ```
   MIGR0344I: Processing configuration file /opt/WebSphere51/AppServer/cloudscape
   /db2j.properties.

   MIGR0344I: Processing configuration file /opt/WebSphere51/AppServer/config/cells
   /migr06/applications/MyBankApp.ear/deployments/MyBankApp/deployment.xml.

   DSRA7600E: Cloudscape migration of database instance /opt/WebSphere61/Express
   /profiles/default/databases/_opt_WebSphere51_AppServer_bin_DefaultDB failed,
   reason: java.sql.SQLException: Failure creating target db

   MIGR0430W: Cloudscape Database /fvt/temp/51BaseXExpress/PostUpgrade50BaseFVTTest9
   /testRun/pre/websphere_backup/bin/DefaultDB failed to migrate <new database name>
   ```

   Important: Call IBM WebSphere Application Server Support if you see a migration failure message for a Cloudscape instance that is accessed by a WebSphere internal component (that is, a component that helps comprise WebSphere Application Server rather than one of your applications).
3. Open the individual database migration log that corresponds with each of your backend Cloudscape databases. These logs have the same timestamp as that of the general post-upgrade log. The logs display more detail about errors that are listed in the general post-upgrade log, as well as expose errors that are not documented by the general log.

   The path name of each database log is WAS_HOME/profiles/*profileName*/logs/*myFulldbPathName*_migrationLog*timestamp*.log.
4. Examine each database migration log for errors. For a completely successful migration, the log displays a message that is similar to the following text:

   ```
   MIGR0429I: Cloudscape Database F:\temp\51BaseXExpress\PostUpgrade50BaseFVTTest2\testRun
   \pre\websphere_backup\bin\DefaultDB was successfully migrated. See log C:\WebSphere61
   \Express\profiles\default\logs\DefaultDB_migrationLogSun-Dec-18-13.31.40-CST-2005.log
   ```

   Otherwise, the log displays error messages in the format of the following example:

   ```
   connecting to source db <jdbc:db2j:/fvt/temp/51BaseXExpress/PostUpgrade50BaseFVTTest9
   /testRun/pre/websphere_backup/bin/DefaultDB>

   connecting to source db <jdbc:db2j:/fvt/temp/51BaseXExpress/PostUpgrade50BaseFVTTest9
   /testRun/pre/websphere_backup/bin/DefaultDB> took 0.26 seconds

   creating target db <jdbc:derby:/opt/WebSphere61/Express/profiles/default/databases
   /_opt_WebSphere51_AppServer_bin_DefaultDB>

    ERROR: An error occurred during migration. See debug.log for more details.
   ```

```
shutting down databases

shutting down databases took   0.055 seconds
```

5. For more data about a migration error, consult the debug log that corresponds with the database migration log. The WebSphere Application Server migration utility triggers a *debug migration trace* by default; this trace function generates the database debug logs. The full path name of a debug log is WAS_HOME/profiles/*profileName*/logs/*myFulldbPathName*_migrationDebug*timestamp*.log.

   The following lines are a sample of debug text. The lines display detailed exception data for the error that is referenced in the previous sample of database migration log data.

```
java.sql.SQLException: Database_opt_WebSphere51_AppServer_bin_DefaultDB already exists. Aborting migration
 at com.ibm.db2j.tools.migration.MigrateFrom51Impl.go(Unknown Source)
 at com.ibm.db2j.tools.migration.MigrateFrom51Impl.doMigrate(Unknown Source)
 at com.ibm.db2j.tools.MigrateFrom51.doMigrate(Unknown Source)
 at com.ibm.ws.adapter.migration.CloudscapeMigrationUtility.migr
```

- The WebSphere Application Server migration utility changes your Cloudscape JDBC configurations whether or not it successfully migrates the database instances that are accessed by your applications. The tool changes Cloudscape JDBC provider class paths, data source implementation classes, and data source helper classes. The following table depicts these changes:

Class type	Old value	New value
JDBC provider class path	${CLOUDSCAPE_JDBC_DRIVER_PATH}/db2j.jar	${DERBY_JDBC_DRIVER_PATH}/derby.jar • Where DERBY_JDBC_DRIVER_PATH is the WebSphere environment variable that defines your Cloudscape JDBC provider • Where derby.jar is the base name of the JDBC driver class file (In your environment, reference the JDBC driver class file by the full path name.)
Data source implementation class: Connection pool	com.ibm.db2j.jdbc.DB2jConnectionPool DataSource	org.apache.derby.jdbc.EmbeddedConnection PoolDataSource
Data source implementation class: XA	com.ibm.db2j.jdbc.DB2jXADataSource	org.apache.derby.jdbc.EmbeddedXADataSource
Data source helper class	com.ibm.websphere.rsadapter.Cloudscape DataStoreHelper	com.ibm.websphere.rsadapter.Derby DataStoreHelper

  Additionally, the db2j.properties file changes:
  - The name WAS_HOME/cloudscape/dbj.properties changes to WAS_HOME/derby/derby.properties
  - Within the file, property names change from db2j.drda.* to derby.drda.*

- A partial or a completely successful database migration changes the location and name of the database according to the following example:
  - Old database name: c:\temp*mydb*
  - New database name: WAS_HOME\profiles*profilename*\databases\c_temp_*mydb*

  Note the exact path names: For both partial and failed migrations, the log messages contain the exact old and new database path names that you must use to run the manual migration. Note these new path names precisely.

If you experience a partial migration, attempt to troubleshoot the new v10.1.x database only if you have expert knowledge of Cloudscape. Otherwise, delete the new database. Perform the manual migration procedure on the original database, just as you do for each database that completely fails auto-migration. Consult “Upgrading Cloudscape manually” for instructions.

Upgrading Cloudscape manually

During the WebSphere Application Server v6.1.x upgrade, the migration tool attempts to upgrade instances of Cloudscape that are accessed through the embedded framework only. (The new version of Cloudscape is version 10.1.x, which is based on Derby.) The automatic upgrade excludes Cloudscape instances that transact with applications through the Network Server framework. This exclusion eliminates the risk of

corrupting third party applications that access the same database instances as WebSphere Application Server. You must manually upgrade database instances that are accessed through the Network Server framework. Do the same for databases that fail the automatic migration.

Do not use Cloudscape v10.1.x as a production database. Use it for development and test purposes only.

Learn more: The new version of Cloudscape combines the Derby runtime with additional benefits, such as IBM Quality Assurance (QA) and national language support (NLS).

- For information about the Cloudscape v10.1.x open source code base, see the Cloudscape section of ibm.com: http://www-306.ibm.com/software/data/cloudscape/.
- You can investigate incompatibilities between Cloudscape v10.1.x and v5.1.60x (plus versions prior to v5.1.60x) by consulting the Cloudscape migration document at http://publib.boulder.ibm.com/epubs/html/c1894711.html.

For instances of Cloudscape that are accessed through the embedded framework, determine which instances completely failed the automatic upgrade process and which ones were only partially upgraded. The article “Verifying the Cloudscape v10.1.x automatic migration” on page 45 documents how to uncover database errors and diagnostic data from various migration logs. The log messages contain the exact old and new database path names that you must use to run the manual migration. Note these new path names precisely.

To minimize the risk of migration errors for databases that were only partially upgraded during the automatic migration process, delete the new database. Troubleshoot the original database according to the log diagnostic data, then perform manual migration on the original database.

The following section consists of steps to migrate Cloudscape instances that are accessed through both frameworks: the embedded as well as the Network Server framework. Steps that apply only to the Cloudscape Network Server framework are marked accordingly. As a migration best practice, ensure that your user ID has one of the following authorities:

- Administrator of the application server that accesses the Cloudscape instance
- A umask that can access the database instance

Otherwise, you might see runtime errors about the database instance being read-only.

1. **Network Server framework only:** Ensure that every client of the Cloudscape database can support Cloudscape v10.1.x. WebSphere Application Server clients of the database must run versions 6.1.x or 6.02.x of Application Server.
2. **Network Server framework only:** Take the database offline. No clients can access it during the migration process.
3. Examine a sample Cloudscape migration script that Application Server provides: either `db2jmigrate.bat`, or `db2jmigrate.sh`. The path of both scripts is `WAS_HOME\derby\bin\embedded\...` . You can modify the script according to the requirements of your environment. Consult the Cloudscape migration document at http://publib.boulder.ibm.com/epubs/html/c1894711.html for information about options that you can use with the script. For example, you can use the option `-DB2j.migrate.ddlFile=`*`filename`* to specify the DDL file for the new database.
4. To generate database debug logs when you run the migration script, ensure that the *debug migration trace* is active. By default, this trace function is enabled. To reactivate the debug trace if it is disabled, set one of the following trace options:
   - all traces*=all
   - com.ibm.ws.migration.WASUpgrade=all
5. Specify your old database name and the full post-migration path of the new database name when you run the script. For example: `E:\WebSphere\AppServer\derby\bin\embedded>db2jMigrate.bat` *`myOldDB myNewDB`* The logs from the automatic migration provide the exact path names to specify for both the old database and the target database. You must use this target database name to specify the new

database, because your migrated Cloudscape data sources (updated by the WebSphere Application Server migration utilities) now point to the target database name. The following sample text demonstrates how log messages display target database names:

```
DSRA7600E: Cloudscape migration of database instance C:\temp\migration2\profiles\AppSrv01\
installedApps\ghongellNode01Cell\DynamicQuery.ear\EmployeeFinderDB to new database instance
C:\WebSphere\AppServer\profiles\AppSrv01\databases\C__WAS602_AppServer_profiles_AppSrv01_
installedApps_ghongellNode01Cell_DynamicQuery.ear_EmployeeFinderDB failed,
reason: java.sql.SQLException: Failure creating target db
```

For instances of Cloudscape that are accessed through the Network Server framework, input any name that you want for the new database. Remember to modify your existing data sources to point to the new database name.

6. When the migration process ends, examine the database migration log to verify the results. The path name of each database migration log is `WAS_HOME/logs/derby/`*`myFulldbPathName`*`_migrationLog.log`.

   For a successful migration, the database migration log displays a message that is similar to the following text:

   ```
   Check E:\WebSphere\AppServer\derby\myOldDB_migrationLog.log for progress
   Migration Completed Successfully
   E:\WebSphere\AppServer\derby\bin\embedded>
   ```

   Otherwise, the log displays error messages in the format of the following example:

   ```
   Check E:\WebSphere\AppServer\derby\myOldDB_migrationLog.log for progress
   ERROR: An error occurred during migration. See debug.log for more details.
   ERROR XMG02: Failure creating target db
   java.sql.SQLException: Failure creating target db
           at com.ibm.db2j.tools.migration.MigrationState.getCurrSQLException(Unknown Source)
           at com.ibm.db2j.tools.migration.MigrateFrom51Impl.handleException(Unknown Source)
           at com.ibm.db2j.tools.migration.MigrateFrom51Impl.go(Unknown Source)
           at com.ibm.db2j.tools.migration.MigrateFrom51Impl.main(Unknown Source)
           at com.ibm.db2j.tools.MigrateFrom51.main(Unknown Source)
   ```

   ...

7. For more data about a migration error, consult the debug log that corresponds with the database migration log. The full path name of a debug log file is `WAS_HOME/logs/derby/`*`myFulldbPathName`*`_migrationDebug.log`

   The following lines are a sample of debug text.

   ```
   sourceDBURL=jdbc:db2j:E:\WebSphere\myOldDB
    newDBURL=jdbc:derby:e:\tempo\myNewDB
    ddlOnly=false
   connecting to source db <jdbc:db2j:E:\WebSphere\myOldDB>
   connecting to source db <jdbc:db2j:E:\WebSphere\myOldDB> took   0.611 seconds
   creating target db <jdbc:derby:e:\tempo\myNewDB>
   creating target db <jdbc:derby:e:\tempo\myNewDB> took   6.589 seconds
   initializing source db data structures
   initializing source db data structures took   0.151 seconds
   recording DDL to create db <E:\WebSphere\myOldDB>
   recording DDL to create db <E:\WebSphere\myOldDB> took   5.808 seconds
   ```

As indicated in the previous steps, the database migration log displays either a `Migration Completed Successfully` message, or a message containing migration failure exceptions.

- For databases that fail migration, troubleshoot according to the logged error data. Then rerun the migration script.
- To access successfully upgraded databases through the embedded framework, modify your data sources to point to the new database names.
- To access successfully upgraded databases through the Network Server framework, you can use either the DB2 Universal JDBC driver or the Derby Client JDBC driver.
  - If you want your existing JDBC configurations to continue to use the DB2 Universal JDBC driver, modify your data sources to point to the new database names.

– If you want to use the Derby Client JDBC driver, which can support XA data sources, modify your JDBC providers to use the new Derby Client JDBC driver class and the new data source implementation classes. Then reconfigure every existing data source to use the correct Derby data source helper class, and to point to the new database name.

  Consult the article in the information center on vendor-specific data sources minimum required settings for all of the new class names.

Security

Migrating, coexisting, and interoperating – Security considerations

Use this topic to migrate the security configuration of previous WebSphere Application Server releases and its applications to the new installation of WebSphere Application Server.

This information addresses the need to migrate your security configurations from a previous release of IBM WebSphere Application Server to WebSphere Application Server Version 6.1 or later. Complete the following steps to migrate your security configurations:

- If security is enabled in the previous release, obtain the administrative server ID and password of the previous release. This information is needed in order to run certain migration jobs.
- You can optionally disable security in the previous release before migrating the installation. No logon is required during the installation.

Follow the steps in Migrating product configurations.

The security configuration of previous WebSphere Application Server releases and its applications are migrated to the new installation of WebSphere Application Server Version 6.1.

If a custom user registry is used in the previous version, the migration process does not migrate the class files that are used by the standalone custom registry in the previous *app_server_root*/`classes` directory. Therefore, after migration, copy your custom user registry implementation classes to the *app_server_root*/`classes` directory.

If you upgrade from WebSphere Application Server, Version 5.x to WebSphere Application Server, Version 6.1, the data that is associated with Version 5.x trust associations is not automatically migrated to Version 6.1. To migrate trust associations, see “Migrating trust association interceptors” on page 54.

If the previous version instance is configured to enable secure connections using digital certificates that are signed by the Digital Certificate Manager (DCM) local certificate authority, those certificates must be renewed. For example, they must be renewed for the previous version instance, the WebSphere Application Server Version 6.1 profile, and all of the Secure Socket Layer-enabled clients and servers that connect to WebSphere Application Server. For more information, see SSL handshake failure using digital certificates signed by a Digital Certificate Manager (DCM) local certificate authority.

i5/OS *SYSTEM certificate stores for applications are deprecated in WebSphere Application Server Version 5. In WebSphere Application Server, you must migrate your applications to use Java keystores.

The os400.security.password.validation.list.object property is profile-dependent. If you are migrating from Version 5, see “Migrating Java thin clients that use the password encoding algorithm” on page 64 for instructions on how to migrate your client configuration.

Interoperating with previous product versions

IBM WebSphere Application Server interoperates with the previous product versions. Use this topic to configure this behavior.

1. Configure WebSphere Application Server Version 6.1 with the same distributed user registry (that is, LDAP or Custom) that is configured with the previous version. Make sure that the same LDAP user registry is shared by all of the product versions.
   a. In the administrative console, select **Security > Secure administration, applications, and infrastructure**.
   b. Choose an available Realm definition and click **Configure**.
   c. Enter a **Primary administrative user name**. This is the identity you use to login to the administrative console or to wsadmin. When interoperating with a previous release, you must use the same server ID. In WebSphere Application Server Version 6.1, there are two choices for server ID. For previous releases, specify a server ID and password. However, if you choose to use the internal server ID, you must be able to override the generated value and to specify the server ID from a previous release.
   d. Click either **Automatically generated server identity** or **Server identity that is stored in the user repository** .
   e. If you select **Automatically generated server identity**, click **Authentication mechanisms and expiration** to change the identity to the one used by the previous release you are interoperating with. Scroll down to the Cross-cell single sign-on section and enter the identity in **Internal server ID**.
   f. If you select **Server identity that is stored in the user repository**, enter the **Server user id** and the associated **Password**.
   g. Fill out the rest of the user registry settings and then click **OK**.
2. Configure the LTPA authentication mechanism. Automatic generation of the LTPA keys should be disabled. If not, keys used by a previous release are lost. Export the current LTPA keys from WebSphere Application Server Version 6.1 and import them into the previous release.
   a. In the administrative console select **Security > Secure administration, applications, and infrastructure**.
   b. Click **Authentication mechanisms and expiration**.
   c. Click the **Key set groups** link, then click the key set group that displays in the Key set groups panel.
   d. Clear the **Automatically generate keys** check box.
   e. Click **OK**, then click **Authentication mechanisms and expiration** in the path at the top of the Key set groups panel.
   f. Scroll down to the Cross-cell single sign-on section, and enter a password to use for encrypting the LTPA keys when adding them to the file.
   g. Enter the password again to confirm the password.
   h. Enter the **Fully qualified key file name** that contains the exported keys.
   i. Click **Export keys**.
   j. Follow the instructions provided in the previous release to import the exported LTPA keys into that configuration.
3. If you are using the default SSL configuration, extract all of the signer certificates from the WebSphere Application Server Version 6.1 common trust store. Otherwise, extract signers where necessary to import them into the previous release.
   a. In the administrative console, click **Security > SSL certificate and key management**.
   b. Click **Key stores and certificates**.
   c. Click **NodeDefaultTrustStore**.
   d. Click **Signer certificates**.
   e. Select one signer and click **Extract**.
   f. Enter a unique path and filename for the signer (for example, `c:\temp\signer1.arm`).
   g. Click **OK**. Repeat for all of the signers in the trust store.

h. Check other trust stores for other signers that might need to be shared with the other server. Repeat steps e through h to extract the other signers.

4. Add the exported signers to `DummyServerTrustFile.jks` and `DummyClientTrustFile.jks` in the `/etc` directory of the back-level product version. If the previous release is not using the dummy certificate, the signer certificate(s) from the previous release must be extracted and added into the WebSphere Application Server Version 6.1 release to enable SSL connectivity in both directions.
   a. Open the key management utility, iKeyman, for that product version.
   b. Start `ikeyman.bat` or `ikeyman.sh` from the `${USER_INSTALL_ROOT}/bin` directory.
   c. Select **Key Database File > Open**.
   d. Open `${USER_INSTALL_ROOT}/etc/DummyServerTrustFile.jks`.
   e. Enter WebAS for the password.
   f. Select **Add** and enter one of the files extracted in step 2. Continue until you have added all of the signers.
   g. Repeat steps c through f for the `DummyClientTrustFile.jks` file.
5. Verify that the application uses the correct Java Naming and Directory Interface (JNDI) name and naming bootstrap port for performing a naming lookup.
6. Stop and restart all of the servers.

Migrating custom user registries

If you built your own custom user registry, consider the migration items listed below. If you have a custom user registry that was provided by a Security Solution Provider, you must contact that provider to ensure that you have the correct version of their custom user registry to support WebSphere Application Server.

In WebSphere Application Server, in addition to the UserRegistry interface, the custom user registry requires the Result object to handle user and group information. This file is already provided in the package and you are expected to use it for the getUsers, getGroups, and the getUsersForGroup methods.

You cannot use other WebSphere Application Server components, for example, data sources, to initialize the custom registry because other components, like the containers, are initialized after security and are not available during the registry initialization. A custom registry implementation is a pure custom implementation, independent of other WebSphere Application Server components.

The getCallerPrincipal enterprise bean method and the getUserPrincipal and getRemoteUser servlet methods return the security name instead of the display name. For more information, see the API documentation.

If the migration tool is used to migrate the WebSphere Application Server Version 5 configuration to WebSphere Application Server Version 6.0.x and later, this migration does not change your existing code. Because the WebSphere Application Server Version 5 custom registry works in WebSphere Application Server Version 6.0.x and later without any changes to the implementation, except when using data sources, you can use the Version 5-based custom registry after the migration without modifying the code.

In WebSphere Application Server Version 6.0.x and later, a case-insensitive authorization can occur when using an enabled custom user registry.

Setting this flag does not have any effect on the user names or passwords. Only the unique IDs that are returned from the registry are changed to lower-case before comparing them with the information in the authorization table, which is also converted to lowercase during runtime.

Before proceeding, look at the UserRegistry interface. See the article in the information center on developing standalone custom registries for a description of each of these methods in detail.

The following steps go through all the changes that are required to move your WebSphere Application Server Version 4.x custom user registry that implemented the old com.ibm.websphere.security.CustomRegistry interface to the com.ibm.websphere.security.UserRegistry interface.

Note: The sample implementation file is used as an **example** when describing the following steps.

1. Change your implementation to `UserRegistry` instead of `CustomRegistry`. Change:

   ```
   public class FileRegistrySample implements CustomRegistry
   ```

   to:

   ```
   public class FileRegistrySample implements UserRegistry
   ```

2. Create the java.rmi.RemoteException exception in the constructors:

   ```
   public FileRegistrySample() throws java.rmi.RemoteException
   ```

3. Change the mapCertificate method to take a certificate chain instead of a single certificate. Change

   ```
   public String mapCertificate(X509Certificate cert)
   ```

   to:

   ```
   public String mapCertificate(X509Certificate[]cert)
   ```

   Having a certificate chain gives you the flexibility to act on the chain instead of one certificate. If you are interested only in the first certificate, take the first certificate in the chain before processing. In WebSphere Application Server Version 6.0.x and later, the mapCertificate method is called to map the user in a certificate to a valid user in the registry when certificates are used for authentication by the Web or the Java clients.

4. Remove the getUsers method.

5. Change the signature of the getUsers(String) method to return a Result object and accept an additional parameter (int). Change:

   ```
   public List getUsers(String pattern)
   ```

   to:

   ```
   public Result getUsers(String pattern, int limit)
   ```

   In your implementation, construct the Result object from the list of the users that is obtained from the user registry (whose number is limited to the value of the limit parameter) and call the setHasMore method on the Result object if the total number of users in the registry exceeds the limit value.

6. Change the signature of the getUsersForGroup(String) method to return a Result object and accept an additional parameter (int) and throw a new exception called NotImplementedException exception. Change the following code:

   ```
   public List getUsersForGroup(String groupName)
            throws CustomRegistryException,
                   EntryNotFoundException {
   ```

   to:

   ```
   public Result getUsersForGroup(String groupSecurityName, int limit)
            throws NotImplementedException,
                   EntryNotFoundException,
                   CustomRegistryException {
   ```

   In WebSphere Application Server Version 6.0.x and later, this method is not called directly by the WebSphere Application Server security component. However, other components of WebSphere Application Server, like the WebSphere Business Integration Server Foundation process choreographer, use this method when staff assignments are modeled using groups. Because this implementation is supported in WebSphere Application Server Version 6.0.x and later, it is recommended that you change the implementation similar to the getUsers method as explained in step 5.

7. Remove the getUniqueUserIds(String) method.
8. Remove the getGroups method.
9. Change the signature of the getGroups(String) method to return a Result object and accept an additional parameter (int). Change the following code:

   ```
   public List getGroups(String pattern)
   ```

   to:

   ```
   public Result getGroups(String pattern, int limit)
   ```

   In your implementation, construct the Result object from the list of the groups that is obtained from the user registry whose number is limited to the value of the limit parameter. Call the setHasMore method on the Result object if the total number of groups in the registry exceeds the limit value.
10. Add the createCredential method. This method is not called at this time, so return as `null`.

    ```
    public com.ibm.websphere.security.cred.WSCredential
       createCredential(String userSecurityName)
             throws CustomRegistryException,
                    NotImplementedException,
                    EntryNotFoundException {
             return null;
       }
    ```

    The first and second lines of the previous code example are split onto two lines for illustrative purposes only.
11. To build the WebSphere Application Server Version 6.0.x and later implementation, make sure you have the `sas.jar` and the `wssec.jar` files in your class path.

    To set the files in your class path, use the following code as a sample and substitute your environment values for the variables that are used in the example:

    ```
    javac -J-Djava.version=1.5 -classpath profile_root/lib/wssec.jar;
    profile_root/lib/sas.jar FileRegistrySample.java
    ```

    Type the previous lines as one continuous line.

    To build the WebSphere Application Server Version 5 custom registry (CustomRegistry) in WebSphere Application Server Version 6.0.x and later, only the `sas.jar` file is required.
12. Copy the implementation classes to the product class path.

    It is recommended that you copy these implementation classes to the *profile_root*`/classes` directory.
13. Use the administrative console to set up the custom registry.

    Follow the instructions in the article in the information center on configuring standalone custom registries to set up the custom registry, including the **Ignore case for authorization** option. Make sure that you add the WAS_UseDisplayName properties if required.

WebSphere Application Server Version 4.x based custom user registry that implemented the old com.ibm.websphere.security.CustomRegistry interface is migrated to the com.ibm.websphere.security.UserRegistry interface.

If you are enabling security, see the article in the information center on enabling security to complete the remaining steps. When completed, save the configuration and restart all the servers. Try accessing some Java 2 Platform, Enterprise Edition (J2EE) resources to verify that the custom registry migration is successful.

Migrating trust association interceptors

Use this topic to manually migrate trust associations.

Note: Data sources are not supported for use within a Trust Association Interceptor (TAI). Data sources are intended for use within J2EE applications and designed to operate within the EJB and Web containers. Trust Association Interceptors do not run within a container, and while data sources may function in the TAI environment, they are untested and not guaranteed to function properly.

The following topics are addressed in this document:
- Changes to the product-provided trust association interceptors
- Migrating product-provided trust association interceptors
- Changes to the custom trust association interceptors
- Migrating custom trust association interceptors

Changes to the product-provided trust association interceptors

For the product-provided implementation for the WebSEAL server, a new optional com.ibm.websphere.security.webseal.ignoreProxy property is added. If this property is set to `true` or `yes`, the implementation does not check for the proxy host names and the proxy ports to match any of the host names and ports that are listed in the com.ibm.websphere.security.webseal.hostnames and the com.ibm.websphere.security.webseal.ports property respectively. For example, if the VIA header contains the following information:

```
HTTP/1.1 Fred (Proxy), 1.1 Sam (Apache/1.1),
HTTP/1.1 webseal1:7002, 1.1 webseal2:7001
```

and the com.ibm.websphere.security.webseal.ignoreProxy property is set to `true` or `yes`, the host name `Fred`, is not used when matching the host names. By default, this property is not set, which implies that any proxy host names and ports that are expected in the VIA header are listed in the host names and the ports properties to satisfy the isTargetInterceptor method.

The previous VIA header information was split onto two lines for illustrative purposes only.

For more information about the com.ibm.websphere.security.webseal.ignoreProxy property, see the article in the information center on configuring single signon using trust association interceptor ++.

Migrating product-provided trust association interceptors

The properties that are located in the `webseal.properties` and `trustedserver.properties` files are not migrated from previous versions of WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server Version 6.0.x using the trust association panels in the administrative console. For more information, see the article in the information center on configuring trust association interceptors.

Changes to the custom trust association interceptors

If the custom interceptor extends the com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor property, implement the following new method to initialize the interceptor:

```
public int init (java.util.Properties props);
```

WebSphere Application Server checks the return status before using the trust association implementation. Zero (0) is the default value for indicating that the interceptor is successfully initialized.

However, if a previous implementation of the trust association interceptor returns a different error status, you can either change your implementation to match the expectations or make one of the following changes:

Method 1:

Add the com.ibm.websphere.security.trustassociation.initStatus property in the trust association interceptor custom properties. Set the property to the value that indicates the interceptor is successfully initialized. All of the other possible values imply failure. In case of failure, the corresponding trust association interceptor is not used.

Method 2:

Add the com.ibm.websphere.security.trustassociation.ignoreInitStatus property in the trust association interceptor custom properties. Set the value of this property to `true`, which tells WebSphere Application Server to ignore the status of this method. If you add this property to the custom properties, WebSphere Application Server does not check the return status, which is similar to previous versions of WebSphere Application Server.

The `public int init (java.util.Properties props` method replaces the `public int init (String propsFile)` method.

The init(Properties) method accepts a java.util.Properties object, which contains the set of properties that is required to initialize the interceptor. All of the properties set for an interceptor are sent to this method. The interceptor can then use these properties to initialize itself. For example, in the product-provided implementation for the WebSEAL server, this method reads the hosts and ports so that a request coming in can be verified to come from trusted hosts and ports. A return value of Zero (0) implies that the interceptor initialization is successful. Any other value implies that the initialization is not successful and the interceptor is not used.

The init(String) method still works if you want to use it instead of implementing the init(Properties) method. The only requirement is that you enter the file name containing the custom trust association properties using the **Custom Properties** link of the interceptor in the administrative console or by using scripts. You can enter the property using either of the following methods. The first method is used for backward compatibility with previous versions of WebSphere Application Server.

Method 1:

The same property names used in the previous release are used to obtain the file name. The file name is obtained by concatenating the `.config` to the `com.ibm.websphere.security.trustassociation.types` property value. If the `myTAI.properties` file is located in the *profile_root*`/properties` directory, set the following properties:

- `com.ibm.websphere.security.trustassociation.types = myTAItype`
- `com.ibm.websphere.security.trustassociation.myTAItype.config =` *profile_root*`/properties/myTAI.properties`

Method 2:

You can set the com.ibm.websphere.security.trustassociation.initPropsFile property in the trust association custom properties to the location of the file. For example, set the following property:

```
com.ibm.websphere.security.trustassociation.initPropsFile=
profile_root/properties/myTAI.properties
```

The previous line of code is split into two lines for illustrative purposes only. Type as one continuous line.

However, it is highly recommended that your implementation be changed to implement the init(Properties) method instead of relying on the init (String propsfile) method.

Migrating custom trust association interceptors

The trust associations from previous versions of WebSphere Application Server are not automatically migrated to WebSphere Application Server Version 6.0.x and later. You can manually migrate these trust associations using the following steps:

1. Recompile the implementation file, if necessary.

For more information, refer to the ″Changes to the custom trust association interceptors″ section previously discussed in this document.

a. Enter `QSH` from a command line to start the QShell environment.
b. Change to the directory that contains your Java source file.
c. Enter the command to recompile the implementation file.

```
javac -Djava.version=1.5 -classpath
app_server_root/lib/wssec.jar:install_root/lib/j2ee.jar .java
```

Where install_root is the installation root.

2. Copy the custom trust association interceptor class files to a location in your product class path. Copy these class files into the *profile_root*/`classes` directory.
3. Start WebSphere Application Server.
4. Enable security to use the trust association interceptor. The properties that are located in your custom trust association properties file and in the `trustedserver.properties` file are not migrated from previous versions of WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server Version 6.0.x and later using the trust association panels in the administrative console.

   For more information, see the article in the information center on configuring trust association interceptors.

Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS)

Use this topic as an example of how to perform programmatic login using the CORBA-based programmatic login APIs.

This document outlines the deprecated Common Object Request Broker Architecture (CORBA) programmatic login APIs and the alternatives that are provided by JAAS. WebSphere Application Server fully supports the Java Authentication and Authorization Service (JAAS) as programmatic login application programming interfaces (API). Refer to the *Securing applications and their environment* PDF for more details on JAAS support.

The following list includes the deprecated CORBA programmatic login APIs.

- `profile_root/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/ServerSideAuthenticator.java`.
- **org.omg.SecurityLevel2.Credentials**. This API is included with the product, but it is not recommended that you use the API.

The APIs that are provided in WebSphere Application Server are a combination of standard JAAS APIs and a product implementation of standard JAAS interfaces.

The following information is only a summary; refer to the JAAS documentation for your platform located at: http://www.ibm.com/developerworks/java/jdk/security/ .

- Programmatic login APIs:
  - javax.security.auth.login.LoginContext
  - javax.security.auth.callback.CallbackHandler interface: The WebSphere Application Server product provides the following implementation of the javax.security.auth.callback.CallbackHandler interface:

    com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl

    Provides a non-prompt CallbackHandler handler when the application pushes basic authentication data (user ID, password, and security realm) or token data to product login modules. This API is recommended for server-side login.

    com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl

    Provides a login prompt CallbackHandler handler to gather basic authentication data (user ID, password, and security realm). This API is recommended for client-side login.

    If this API is used on the server side, the server is blocked for input.

– javax.security.auth.callback.Callback interface:
  javax.security.auth.callback.NameCallback
  Provided by JAAS to pass the user name to the LoginModules interface.
  javax.security.auth.callback.PasswordCallback
  Provided by JAAS to pass the password to the LoginModules interface.
  com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl
  Provided by the product to perform a token-based login. With this API, an application can pass a token-byte array to the LoginModules interface.
– **javax.security.auth.spi.LoginModule interface**
  WebSphere Application Server provides a LoginModules implementation for client and server-side login. Refer to the *Securing applications and their environment* PDF for details.

- javax.security.Subject:
  com.ibm.websphere.security.auth.WSSubject
  An extension provided by the product to invoke remote J2EE resources using the credentials in the javax.security.Subject
  com.ibm.websphere.security.cred.WSCredential
  After a successful JAAS login with the WebSphere Application Server LoginModules interfaces, a com.ibm.websphere.security.cred.WSCredential credential is created and stored in the Subject.
  com.ibm.websphere.security.auth.WSPrincipal
  An authenticated principal that is created and stored in a Subject that is authenticated by the WebSphere Application Server LoginModules interface.

1. Use the following as an example of how to perform programmatic login using the CORBA-based programmatic login APIs: The CORBA-based programmatic login APIs are replaced by JAAS login.

   Note: The LoginHelper application programming interface (API) that is used in the following example is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release. It is recommended that you use the JAAS programmatic login APIs that are shown in the next step.

```
public class TestClient {
...
private void performLogin() {
// Get the ID and password of the user.
String userid = customGetUserid();
String password = customGetPassword();

// Create a new security context to hold authentication data.
LoginHelper loginHelper = new LoginHelper();
try {
// Provide the ID and password of the user for authentication.
org.omg.SecurityLevel2.Credentials credentials =
loginHelper.login(userid, password);

// Use the new credentials for all future invocations.
loginHelper.setInvocationCredentials(credentials);
// Retrieve the name of the user from the credentials
// so we can tell the user that login succeeded.

String username = loginHelper.getUserName(credentials);
System.out.println("Security context set for user: "+username);
} catch (org.omg.SecurityLevel2.LoginFailed e) {
// Handle the LoginFailed exception.
}
```

```
}
...
}
```

2. Use the following example to migrate the CORBA-based programmatic login APIs to the JAAS programmatic login APIs.

   The following example assumes that the application code is granted for the required Java 2 security permissions. For more information, see the *Securing applications and their environment* PDF and the JAAS documentation located at http://www.ibm.com/developerworks/java/jdk/security/.

```
public class TestClient {
...
private void performLogin() {
// Create a new JAAS LoginContext.
javax.security.auth.login.LoginContext lc = null;

try {
// Use GUI prompt to gather the BasicAuth data.
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by  login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS Login Configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is an protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception: "
+ e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
```

```
// Retrieve the name of the principal from the Subject
// so we can tell the user that login succeeded,
// should only be one WSPrincipal.
java.util.Set ps =
s.getPrincipals(com.ibm.websphere.security.auth.WSPrincipal.class);
java.util.Iterator it = ps.iterator();
while (it.hasNext()) {
com.ibm.websphere.security.auth.WSPrincipal p =
(com.ibm.websphere.security.auth.WSPrincipal) it.next();
System.out.println("Principal: " + p.getName());
}
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
}
...
}
```

Migrating from the CustomLoginServlet class to servlet filters

Use this topic to allow migration in an application that uses form-based login and servlet filters without the use of the CustomLoginServlet class.

The CustomLoginServlet class is deprecated in WebSphere Application Server Version 5. Those applications using the CustomLoginServlet class to perform authentication now need to use form-based login. Using the form-based login mechanism, you can control the look and feel of the login screen. In form-based login, a login page is specified and displays when retrieving the user ID and password information. You also can specify an error page that displays when authentication fails.

If login and error pages are not enough to implement the CustomLoginServlet class, use servlet filters. Servlet filters can dynamically intercept requests and responses to transform or use the information that is contained in the requests or responses. One or more servlet filters attach to a servlet or a group of servlets. Servlet filters also can attach to JavaServer Pages (JSP) files and HTML pages. All the attached servlet filters are called before invoking the servlet.

Both form-based login and servlet filters are supported by any Servlet 2.3 specification-compliant Web container. A form login servlet performs the authentication and servlet filters can perform additional authentication, auditing, or logging tasks.

To perform pre-login and post-login actions using servlet filters, configure these servlet filters for either form login page or for /j_security_check URL. The j_security_check is posted by the form login page with the j_username parameter that contains the user name and the j_password parameter that contains the password. A servlet filter can use user name and password information to perform more authentication or meet other special needs.

1. Develop a form login page and error page for the application.
   Refer to the *Securing applications and their environment* PDF for details.
2. Configure the form login page and the error page for the application as described in .
   Refer to the *Securing applications and their environment* PDF for details.
3. Develop servlet filters if additional processing is required before and after form login authentication.
   Refer to the *Securing applications and their environment* PDF for details.
4. Configure the servlet filters that are developed in the previous step for either the form login page URL or for the /j_security_check URL. Use an assembly tool or development tools like Rational Application

Developer to configure filters. After configuring the servlet filters, the `web-xml` file contains two stanzas. The first stanza contains the servlet filter configuration, the servlet filter, and its implementation class. The second stanza contains the filter mapping section and a mapping of the servlet filter to the URL.

For more information, see the *Securing applications and their environment* PDF.

This migration results in an application that uses form-based login and servlet filters without the use of the CustomLoginServlet class.

The new application uses form-based login and servlet filters to replace the CustomLoginServlet class. Servlet filters also are used to perform additional authentication, auditing, and logging.

Migrating Java 2 security policy

Use this topic for guidance pertaining to migrating Java 2 security policy.

Previous WebSphere Application Server releases

WebSphere Application Server uses the Java 2 security manager in the server runtime to prevent enterprise applications from calling the System.exit and the System.setSecurityManager methods. These two Java application programming interfaces (API) have undesirable consequences if called by enterprise applications. The System.exit API, for example, causes the Java virtual machine (application server process) to exit prematurely, which is not a beneficial operation for an application server.

To support Java 2 security properly, all the server runtime must be marked as privileged (with doPrivileged API calls inserted in the correct places), and identify the default permission sets or policy. Application code is not privileged and subject to the permissions that are defined in the policy files. The doPrivileged instrumentation is important and necessary to support Java 2 security. Without it, the application code must be granted the permissions that are required by the server runtime. This situation is due to the design and algorithm that is used by Java 2 security to enforce permission checks. Refer to the Java 2 security check permission algorithm.

The following two permissions are enforced by the Java 2 security manager (hard coded) for WebSphere Application Server:

- java.lang.RuntimePermission(exitVM)
- java.lang.RuntimePermission(setSecurityManager)

Application code is denied access to these permissions regardless of what is in the Java 2 security policy. However, the server runtime is granted these permissions. All the other permission checks are not enforced.

Only two permissions are supported:

- java.net.SocketPermission
- java.net.NetPermission

However, not all the product server runtime is properly marked as privileged. You must grant the application code all the other permissions besides the two listed previously or the enterprise application can potentially fail to run. This Java 2 security policy for enterprise applications is liberal.

What changed

Java 2 Security is fully supported in WebSphere Application Server, which means all permissions are enforced. The default Java 2 security policy for enterprise application is the recommended permission set defined by the Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 specification. Refer to the *profile_root*/config/cells/*cell_name*/nodes/*node_name*/app.policy file for the default Java 2 security policy granted to enterprise applications. This policy is much more stringent compared to previous releases.

All policy is declarative. The product security manager honors all policy declared in the policy files. There is an exception to this rule: enterprise applications are denied access to permissions declared in the *profile_root*/config/cells/*cell_name*/filter.policy file.

Note: The default Java 2 security policy for enterprise applications is much more stringent and all the permissions are enforced in WebSphere Application Server Version 6.0.x and later. The security policy might fail because the application code does not have the necessary permissions granted where system resources, such as file I/O, can be programmatically accessed and are now subject to the permission checking.

In application code, do not use the setSecurityManager permission to set a security manager. When an application uses the setSecurityManager permission, there is a conflict with the internal security manager within WebSphere Application Server. If you must set a security manager in an application for RMI purposes, you also must enable the **Use Java 2 security to restrict application access to local resources** option on the Secure administration, applications, and infrastructure page within the WebSphere Application Server administrative console. WebSphere Application Server then registers a security manager. The application code can verify that this security manager is registered by using System.getSecurityManager() application programming interface (API).

Migrating system properties

The following system properties are used in previous releases in relation to Java 2 security:

- **java.security.policy**. The absolute path of the policy file (action required). This system property contains both system permissions (permissions granted to the Java virtual machine (JVM) and the product server runtime) and enterprise application permissions. Migrate the Java 2 security policy of the enterprise application to WebSphere Application Server Version 6.0.x. For Java 2 security policy migration, see the steps for migrating Java 2 security policy.
- **enableJava2Security**. Used to enable Java 2 security enforcement (no action required). This system property is deprecated; a flag in the WebSphere configuration application programming interface (API) is used to control whether to enabled Java 2 security. Enable this option through the administrative console.
- **was.home**. Expanded to the installation directory of WebSphere Application Server (action might be required). This system property is deprecated; superseded by the ${user.install.root} and ${was.install.root} properties. If the directory contains instance-specific data then ${user.install.root} is used; otherwise ${was.install.root} is used. Use these properties interchangeably for the WebSphere Application Server or the Network Deployment environments. See the steps for migrating Java 2 security policy.

Migrating the Java 2 Security Policy

No easy way exists to migrate the Java policy file to WebSphere Application Server Version 6.0.x and later automatically because of a mixture of system permissions and application permissions in the same policy file. Manually copy the Java 2 security policy for enterprise applications to a was.policy or app.policy file. However, migrating the Java 2 security policy to a was.policy file is preferable because symbols or relative code base is used instead of an absolute code base. This process has many advantages. Grant the permissions that are defined in the was.policy to the specific enterprise application only, while permissions in the app.policy file apply to all the enterprise applications that run on the node where the app.policy file belongs.

Refer to the *Securing applications and their environment* PDF for more details on policy management.

The following example illustrates the migration of a Java 2 security policy from a previous release. The contents include the Java 2 security policy file for the app1.ear enterprise application and the system permissions, which are permissions that are granted to the Java virtual machine (JVM) and the product server runtime.

The default location for the Java 2 security policy file is *profile_root*/properties/java.policy. Default permissions are omitted for clarity:

```
// For product Samples
   grant codeBase "file:${app_server_root}/installedApps/app1.ear/-" {
     permission java.security.SecurityPermission "printIdentity";
     permission java.io.FilePermission "${app_server_root}${/}temp${/}somefile.txt",
       "read";
   };
```

For clarity of illustration, all the permissions are migrated as the application level permissions in this example. However, you can grant permissions at a more granular level at the component level (Web, enterprise beans, connector or utility Java archive (JAR) component level) or you can grant permissions to a particular component.

1. Ensure that Java 2 security is disabled on the application server.
2. Create a new was.policy file, if the file is not present, or update the was.policy file for migrated applications in the configuration repository with the following contents:

   ```
   grant  codeBase  ″file:${application}″  {
          permission  java.security.SecurityPermission  ″printIdentity″;
          permission  java.io.FilePermission  ″
                      ${user.install.root}${/}temp${/}somefile.txt″,  ″read″;
       };
   ```

   The third and fourth lines in the previous code sample are presented on two lines for illustrative purposes only.

   The was.policy file is located in the *profile_root*/config/cells/*cell_name*/applications/app.ear/deployments/app/META-INF/ directory.
3. Use an assembly tool to attach the was.policy file to the enterprise archive (EAR) file.

   You also can use an assembly tool to validate the contents of the was.policy file. For more information, see the *Securing applications and their environment* PDF.
4. Validate that the enterprise application does not require additional permissions to the migrated Java 2 security permissions and the default permissions set declared in the ${user.install.root}/config/cells/*cell_name*/nodes/*node_name*/app.policy file. This validation requires code review, code inspection, application documentation review, and sandbox testing of migrated enterprise applications with Java 2 security enabled in a preproduction environment. Refer to developer kit APIs protected by Java 2 security for information about which APIs are protected by Java 2 security. If you use third-party libraries, consult the vendor documentation for APIs that are protected by Java 2 security. Verify that the application is granted all the required permissions, or it might fail to run when Java 2 security is enabled.
5. Perform preproduction testing of the migrated enterprise application with Java 2 security enabled. Enable trace for the WebSphere Application Server Java 2 security manager in a preproduction testing environment with the following trace string: com.ibm.ws.security.core.SecurityManager=all=enabled. This trace function can be helpful in debugging the AccessControlException exception that is created when an application is not granted the required permission or some system code is not properly marked as privileged. The trace dumps the stack trace and permissions that are granted to the classes on the call stack when the exception is created.

   For more information, see the *Securing applications and their environment* PDF.

   Note: Because the Java 2 security policy is much more stringent compared with previous releases, the administrator or deployer must review their enterprise applications to see if extra permissions are required before enabling Java 2 security. If the enterprise applications are not granted the required permissions, they fail to run.

Migrating Java thin clients that use the password encoding algorithm

To migrate Java thin clients that are enabled for OS400 password encoding, use the following information to modify the Java client invocation so that the os400.security.password properties are no longer set on the invocation.

The password encoding feature offers the following encoding algorithms:

- XOR, which is the default
- OS400

In Version 5 and later, the value of the os400.security.password.validation.list.object property is dependant upon the property value passed to the thin client using the JAVA_FLAGS environment variable. The JAVA_FLAGS environment variable is set by the setupClient script. The setupClient script calls the setupCmdLine script, which is where the value for the os400.security.password.validation.list.object property is set. For example, if a Version 6.x Base Edition Java client is passed -profileName default, then the setupClient script calls the *profile_root*/default/bin/setupCmdLine file.

To migrate Java thin clients that are enabled for OS400 password encoding, modify the Java client invocation so that the os400.security.password properties are no longer set on the invocation. The following code sample does not contain the os400.security.password properties:

```
java -classpath $MY_CLIENT_CLASSES:app_server_root/classes/wsa400.jar:$WAS_CLASSPATH \
  $CLIENTSAS $JAVA_FLAGS \
  -Djava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory \
  -Djava.naming.provider.url=iiop://server1:10151 \
  MyClientClass $*
```

Perform the following steps if the following condition is true:

- If the passwords in the sas.client.props file for that profile are encoded with the OS400 password encoding algorithm

1. Replace all of the OS400 encoded passwords, which have {OS400}) prefixes in the sas.client.props file for the Application Server profile, with the clear text values of the passwords.
2. Encode the passwords using the PropFilePasswordEncoder Qshell command.

   For more information, see the *Securing applications and their environment* PDF.

Attention: You can configure a WebSphere Application Server profile to encode passwords with the XOR algorithm even though the profile is enabled to decode passwords that were encoded with either the OS400 algorithm or the XOR algorithm. If you encode these passwords with the XOR algorithm, then the passwords in the sas.client.props file are encoded with the XOR algorithm.

Enabling embedded Tivoli Access Manager

Embedded Tivoli Access Manager is not enabled by default, and you need to configure it for use.

Enabling Tivoli Access Manager security within WebSphere Application Server requires:

- A supported Lightweight Directory Access Protocol (LDAP) installed somewhere on your network. This user registry contains the user and group information for both Tivoli Access Manager and WebSphere Application Server.
- Tivoli Access Manager server exists and is configured to use the user registry. For details on the installation and configuration of Tivoli Access Manager, refer to the IBM Tivoli Access Manager for e-business information center.

  Note: WebSphere Application Server contains an embedded client for Tivoli Access Manager. To use Tivoli Access Manager, you must also configure the Tivoli Access Manager server.

However, the server version must be the same version or later as the client version. For information on the supported version of Tivoli Access Manager, see WebSphere Application Server - Supported Prerequisites.

- WebSphere Application Server is installed either in a single server model or as WebSphere Application Server Network Deployment.
- When administrative security is configured with a Federal Information Processing Standard (FIPS) provider, the Tivoli Access Manager server must be configured for FIPS as well

Complete the following steps to enable embedded Tivoli Access Manager security:

1. Create the security administrative user.
   For more information, see the *Securing applications and their environment* PDF.
2. Configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager .
   For more information, see the *Securing applications and their environment* PDF.
3. Enable WebSphere Application Server security. When you are using Tivoli Access Manager you must configure LDAP as the user registry.
   For more information, see the *Securing applications and their environment* PDF.
4. Enable the JACC provider for Tivoli Access Manager.
   For more information, see the *Securing applications and their environment* PDF.

Propagating security policy of installed applications to a JACC provider using wsadmin scripting

It is possible that you have applications installed prior to enabling the Java Authorization Contract for Containers (JACC)-based authorization. You can start with default authorization and then move to an external provider-based authorization using JACC later.

Also, during application installation or modification you might have had problems propagating the security policy information to the JACC provider. For example, network problems might occur, the JACC provider might not be available, and so on. For these cases, the security policy of the previously installed applications does not exist in the JACC provider to make the access decisions. One choice is to reinstall the applications involved. However, you can avoid reinstalling by using the wsadmin scripting tool. Use this tool to propagate information to the JACC provider independent of the application installation process. The tool eliminates the need for reinstalling the applications.

The tool uses the SecurityAdmin MBean to propagate the policy information in the deployment descriptor of any installed application to the JACC provider. You can invoke this tool using wsadmin at the base application server for base and deployment manager level for Network Deployment. Note that the SecurityAdmin MBean is available only when the server is running.

Use `propagatePolicyToJACCProvider(String appNames)` to propagate the policy information in the deployment descriptor of the enterprise archive (EAR) files to the JACC provider. If the RoleConfigurationFactory and the RoleConfiguration interfaces are implemented by the JACC provider, the authorization table information in the binding file of the EAR files is also propagated to the provider. See the *Securing applications and their environment* PDF for more information about these interfaces.

The appNames String contains the list of application names, delimited by a colon (:), whose policy information must be stored in the provider. If a null value is passed, the policy information of the deployed applications is propagated to the provider.

Also, be aware of the following items:

- Before migrating applications to the Tivoli Access Manager JACC provider, create or import the users and groups that are in the applications to Tivoli Access Manager.

- Depending on the application or the number of applications that are propagated, you might have to increase the request time-out period either in the `soap.client.props` file in the directory *profile_root*`/properties` (if using SOAP) or in the `sas.client.props` file (if using RMI) for the command to complete. You can set the request time-out value to 0 to avoid the timeout problem, and change it back to the original value after the command is run.

1. Configure your JACC provider in WebSphere Application Server.

   See the *Securing applications and their environment* PDF for more information.
2. Restart the server.
3. Enter the following commands:

```
// use the SecurityAdmin MBean at the Deployment Manager or the unmanaged base application
// server connect to the appropriate process (Deployment Manager or base application server)
wsadmin -user serverID -password serverPWD
// To get the SecurityAdmin MBean for Deployment Manager
wsadmin> set secadm [$AdminControl queryNames type=SecurityAdmin,process=dmgr,*]
// or to get the SecurityAdmin MBean for a unmanaged base application server (replace the
// process name to match your configuration)
wsadmin> set secadm [$AdminControl queryNames type=SecurityAdmin,process=server1,*]
// to propagate specific applications security policy information
wsadmin>set appNames [list app1:app2]
// or to propagate all applications installed
wsadmin>set appNames [list null]

// Run the command to propagate
wsadmin>$AdminControl invoke $secadm propagatePolicyToJACCProvider $appNames
```

Naming and directory

JNDI interoperability considerations

You must take extra steps to enable your programs to interoperate with non-WebSphere Application Server JNDI clients and to bind resources from MQSeries to a name space.

EJB clients running in an environment other than WebSphere Application Server accessing EJB applications running on WebSphere Application Server V5 or V6 servers

When an EJB application running in WebSphere Application Server V5 or V6 is accessed by a non-WebSphere Application Server EJB client, the JNDI initial context factory is presumed to be a non-WebSphere Application Server implementation. In this case, the default initial context is the cell root. If the JNDI service provider being used supports CORBA object URLs, the corbaname format can be used to look up the EJB home. The construction of the stringified name depends on whether the object is installed on a single server or cluster.

Single server

Following is a URL that has the bootstrap host **myHost**, the port **2809**, and the enterprise bean installed in the server **server1** in node **node1** and bound in that server under the name **myEJB**:

```
initialContext.lookup(
  "corbaname:iiop:myHost:2809#cell/nodes/node1/servers/server1/myEJB");
```

Server cluster

Following is a URL that has the bootstrap host **myHost**, the port **2809**, and the enterprise bean installed in a server cluster named **myCluster** and bound in that cluster under the name **myEJB**:

```
initialContext.lookup(
    "corbaname:iiop:myHost:2809#cell/clusters/myCluster/myEJB");
```

The lookup works with any name server bootstrap host and port configured in the same cell.

The lookup also works if the bootstrap host and port belong to a member of the cluster itself. To avoid a single point of failure, the bootstrap server host and port for each cluster member can be listed in the URL as follows:

```
initialContext.lookup(
    "corbaname:iiop:host1:9810,:host2:9810#cell/clusters/myCluster/myEJB");
```

The name prefix **cell/clusters/myCluster/** is not necessary if boostrapping to the cluster itself, but it will work. The prefix is needed, however, when looking up enterprise beans in other clusters. Name bindings under the **clusters** context are implemented on the name server to resolve to the server root of a running cluster member during a lookup; thus avoiding a single point of failure.

Without CORBA object URL support

If the JNDI initial context factory being used does not support CORBA object URLs, the initial context can be obtained from the server, and the lookup can be performed on the initial context as follows:

```
Hashtable env = new Hashtable();
env.put(CONTEXT.PROVIDER_URL, "iiop://myHost:2809");
Context ic = new InitialContext(env);
Object o = ic.lookup("cell/clusters/myCluster/myEJB");
```

Binding resources from MQSeries 5.2

In releases previous to WebSphere Application Server V5, the MQSeries jmsadmin tool could be used to bind resources to the name space. When used with a WebSphere Application Server V5 or V6 name space, the resource is bound within a transient partition in the name space and does not persist past the life of the server process. Instead of binding the MQSeries resources with the jmsadmin tool, bind them from the WebSphere Application Server administrative console, under **Resources** in the console navigation tree.

Application profiling

Running Version 5 Application Profiles on Version 6

Java 2 platform, Enterprise Edition (J2EE) 1.3 applications created using WebSphere Application Server Version 5.x have an application profile configuration formatted for Version 5.x. Although you can use applications with an application profile configuration from Version 5.x on WebSphere Application Server Version 6.x, you must change a setting. Also, there are several implications to using version 5.x application profiles on version 6.x.

The WebSphere Application Server application profiling function works under the *unit of work* concept. This gives it a more predictable data access pattern based on the active unit of work, which could be either a transaction or an ActivitySession. See the *Developing and deploying applications* PDF for more information.

- Set the Application Profile service on your server to enable the Application Profiling 5.x Compatibility Mode as the default.

  See the *Developing and deploying applications* PDF.

  This setting is necessary to support Java 2 platform, Enterprise Edition (J2EE) 1.3 applications with an application profile configuration from WebSphere Application Server Version 5.x. The 5.x compatibility mode has a fair amount of performance overhead on a Version 6 server. Because of this, if there is no J2EE 1.3 application with an application profile V5.x configuration installed, the server *does not load* the support for the 5.x compatibility mode during startup, even when the 5.x compatibility mode is turned on.

  After the server starts without loading the 5.x compatibility mode support, if a J2EE 1.3 application with an application profile V5.x configuration installs on the server and attempts to start, the following message is displayed, and the server must be restarted:

ACIN0031E: The J2EE 1.3 application <ApplicationName> is configured for application profiling and is installed and starting on a running server that enables Application Profiling 5.x Compatibility Mode. You must re-start the server.

This situation only happens when:

1. the server started with the Application Profile service enabled and 5.x compatibility mode turned on, but no J2EE 1.3 applications were installed at server start up. Therefore, the server run time automatically ignores the 5.x compatibility in order to avoid performance costs associated with it.
2. you try to install and start a J2EE 1.3 application with an application profile configured in Version 5.x, but the 5.x compatibility mode is turned off.

To avoid this situation, you must install at least one J2EE 1.3 application with an application profile Version 5.x configuration *before* starting the server, or restart the server after installing a J2EE 1.3 application with the application profile configured in Version 5.x.

- Ideally, upgrade your J2EE 1.3 applications to use the Version 6.x application profiling configuration and turn off the Application Profiling 5.x Compatibility Mode through the administrative console.

  See the *Developing and deploying applications* PDF.
- Migrate any application you have configured with application profiling in Version 5.

  Application profiles migration requires you to re-configure your applications in the Application Server Toolkit (AST). See the *Developing and deploying applications* PDF for more information.
- Rework the usage of the TaskNameManager API if it is used in your applications. The TaskNameManager API is not supported in container-managed transaction beans, and the setTaskName() method must be called before beginning a new unit of work.

  See the *Developing and deploying applications* PDF for more information.

Migrating Version 5 Application Profiles to Version 6

You can migrate an application with a WebSphere Application Server Version 5.x application profile to a version 6 application profile. You might want to migrate because the application profiling function works under a different concept in version 6 and a major difference exists between Version 5.x and Version 6 in the scoping of a task name.

For the implications of using Java 2 platform, Enterprise Edition (J2EE) 1.3 applications with an application profile configuration from WebSphere Application Server Version 5.x on Version 6, see “Running Version 5 Application Profiles on Version 6” on page 67.

In Version 5.x, a task name could be assigned to a bean, web, or client method. When a method was invoked that had a task name configured on it, either that task name became the active task name (known as ’run as specified’) or the caller’s task name, should one exist, became the active task name (known as ’run as caller’). In Version 6, the application profiling function works under the unit of work (UOW) concept. UOW in this case means either a transaction or an ActivitySession. This means that the task name on a method is used only when a UOW is begun, because of that method being invoked. This gives it a more predictable data access pattern based on the active unit of work. To be more specific, this approach ensures that a bean type with only one configured access intent is loaded within a UOW, because a bean is configured with only one access intent within an application profile. This configured access intent for a bean type is determined at assembly time and is enforced by the Application Profile service. In Version 5.x, a bean type could be loaded in different access intents and cause deadlock problems within a UOW.

A major difference between Version 5.x and Version 6 is with the scoping of a task name. In Version 5.x, a task is not necessarily associated with a UOW. In Version 6, a task name is always associated with a UOW and that task name does not change for the duration of that UOW. When a UOW associated with a method is begun because of that method being invoked, if a task name is associated with the method then that task name is used to name the UOW. A task assigned to a UOW is considered a named UOW. If a task name is not associated with the method that began the UOW, then a default access intent is used and the UOW is unnamed. A UOW can only be named when the UOW is begun. Any task names associated with a method are ignored if that method does not begin a UOW (either container managed or component managed).

To migrate a Version 5.x application to use version 6 application profile functionality do the following:

1. Configure a UOW on a method.

   In Version 5.x it wasn't necessary to associate a task with a UOW. In Version 6, a task can only be associated with a UOW when that UOW is begun. Therefore, the first step to convert a Version 5.x application to Version 6 is to define a UOW on any method that has a task name (and eventually an Application Profile) associated with it. A UOW can either be container managed or component managed and can either be a transaction or an ActivitySession.

2. Associate a method level task name with the same methods used to define the UOW you just created.

   When a method is called that causes a UOW to begin, if a task name is associated with the same method then that task name becomes the active task name and is associated with the UOW. Therefore, a task name needs to be defined on a method that participates in a UOW and that needs an Application Profile associated with it. A task can be associated with a method either declaratively through an application's deployment descriptor or programmatically through the TaskNameManager API.

   The *Developing and deploying applications* PDF describes how to configure task names either declaratively or programmatically.

3. Reassign 'run as caller' tasks. In Version 5.x, you could define a container task to be either 'run as caller' or 'run as specified'. In the 'run as caller case', if the configured methods are invoked with an associated task, the method is executed with the imported (caller's) task. If a method is invoked by a request that is not associated with a task, the method continues to execute without a task. In the 'run as specified' case, the configured methods are never invoked with an imported (caller's) task. Instead, the method executes as the specified task name.

   In Version 6, these two ″run as″ options have explicitly been removed yet are still implied. In other words, when a user defines a method with a task name, the user is specifying a task name to be used when the method is invoked. This is similar to 'run as specified' in Version 5.x. However, it is **not exactly** 'run as specified' because in Version 6 a task name can only be associated with a UOW when that UOW is begun. In other words, even if a method has a task name associated with it in Version 6, that task name is ignored if there is already a UOW, or if the method does not cause a UOW to begin. In Version 5.x, a 'run as specified' task defined on a method always became the active task name when the method was invoked.

   The 'run as caller' option of Version 5.x is also implicitly available in Version 6, although not explicitly stated. When a method (callee) is invoked by another method (caller) that has a UOW associated with it, the callee is associated with the caller's UOW if the callee does not begin a new UOW. In other words, the callee runs with the caller's task. The name of the UOW is the task name associated with the callee, even if the callee has its own task name specified. Keep in mind that the UOW could be unnamed, in which case no task name is associated with the caller or callee.

4. **Optional:** Remove method level access intent. In Version 6.0, method level access intent has been deprecated. It is suggested that you remove method level access intent from your applications. In its place assign task names to a method and then assign the task name to an Application Profile with the necessary access intent.

Application profiling interoperability

Using application profiling with 5.x compatibility mode or in a clustered environment with mixed WebSphere Application Server product versions and mixed platforms can affect its behavior in different ways.

The effect of 5.x Compatibility Mode

Application profiling supports *forward* compatibility. Application profiles created in previous versions of WebSphere Application Server (Enterprise Edition 5.0 or WebSphere Business Integration Server Foundation 5.1) can only run in WebSphere Application Server Version 6 or later if the Application Profiling 5.x Compatibility Mode attribute is turned on. If the 5.x Compatibility Mode attribute is off, Version 5 application profiles might display unexpected behavior.

Similarly, application profiles that you create using the latest version of WebSphere Application Server are not compatible with Version 5 or earlier versions. Even applications configured with application profiles run on Version 6.x servers with the Application Profiling 5.x Compatibility Mode attribute turned on cannot interact with applications configured with profiles run on Version 5 servers.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to **true** in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the *launchClient* command.

Considerations for a clustered environment

In a clustered environment with mixed WebSphere Application Server product versions and mixed platforms, applications configured with application profiles might exhibit unexpected behavior because previous versions of server members cannot support the application profiling of Version 6.x.

If a clustered environment contains both Version 5.x and 6.x server members, and if any applications are configured with application profiles, the Application Profiling 5.x Compatibility Mode attribute must be turned on in Version 6 server members. Still, this cluster can only support Version 5 application profiling behavior. To support applications configured with Version 6 application profiles in a cluster environment, all server members in the cluster must be at the Version 6.x level.

WebSphere Application Server Enterprise Edition Version 5.0.2

If you use WebSphere Application Server Enterprise Edition 5.0.2, you must apply WebSphere Application Server Version 5 service pack 7 or later service pack to enable Application Profiling interoperability.

Asynchronous beans

Interoperating with asynchronous beans

Asynchronous beans support Serialized WorkWithExecutionContext interoperability with objects serialized in 5.0.2 or later.

For more information on migrating to WebSphere Application Server Version 6 from previous product releases, see the topic, Chapter 5, "Migrating product configurations," on page 77.

1. Install the Version 6.x product. Installing the product creates a stand-alone application server.
2. Migrate the previous release to the Version 6.1 product.

   See "Overview of migration, coexistence, and interoperability" on page 72.

Chapter 4. Migrating and coexisting

Migrating involves collecting the configuration information from a previous release of a WebSphere Application Server product and merging it into a configuration for a new release. Coexisting involves running a new release of a WebSphere Application Server product on the same machine at the same time as you run an earlier release or running two installations of the same release of a WebSphere Application Server product on the same machine at the same time.

See “Overview of migration, coexistence, and interoperability” on page 72 and “Premigration considerations” on page 73.

The migration tools basically save the existing WebSphere configurations and user applications in a backup directory and then process the contents of this backup directory to migrate the configurations and your applications from previous WebSphere Application Server releases to the latest release.

If you have a previous version of WebSphere Application Server, you must decide whether to copy the configuration and applications of the previous version to the new version. Migration does not uninstall the previous version. To run an earlier release and the new release at the same time is coexistence. To support coexistence, you need to provide non-default port assignments during profile creation.

Note: For migration scenarios involving the possibility of rolling back to the previous version, you can choose to have the same port definitions and run either one version or the other.

For help in troubleshooting problems when migrating, see Chapter 11, “Troubleshooting migration,” on page 115.

You can coexist with or migrate the applications and configuration from a previous version of WebSphere Application Server.

For information on migrating to Version 6.1, see Chapter 5, “Migrating product configurations,” on page 77. For more information on coexistence among releases, see “Coexistence support” on page 105.

1. Prepare to migrate or update product prerequisites and corequisites to supported versions.

   Refer to the IBM WebSphere Application Server supported hardware, software, and APIs site for current requirements.
2. Install the WebSphere Application Server Version 6.1 product.
3. Migrate your WebSphere Application Server Version 5.x or Version 6.0.x product configuration to Version 6.1.

   You have the choice between migrating your configuration automatically using the migration tools or manually.
   - Use the migration tools to automatically migrate your configuration.

     See “Using the migration tools to migrate product configurations” on page 87.

     Consider the following points related to using the migration tools to automatically migrate your configuration:
     - Advantages
       - You copy the old configuration automatically.

         This includes all resource definitions, virtual host definitions, security settings, and so forth.
       - You recreate the same exact Version 5.x or 6.0.x configuration in Version 6.1, including the server definitions and deployed applications by default.
       - You can enable support for script compatibility.

         See “WASPostUpgrade command” on page 91.
     - Consideration

This approach will copy the configuration exactly. A different approach, such as the manual approach, is required if your existing configuration is different from that which you want for the new configuration.

- Manually migrate your configuration.

  Migrating your configuration manually involves the following activities:

  – You start with a clean slate and build up a new environment for Version 6.1.

  – Ideally, you would use an existing set of administration scripts to set up the complete Version 6.1 environment.

  – You move your applications to Version 6.1 as they are tested on Version 6.1.

  Consider the following points related to manually migrating your configuration:

  – Advantages

    - You can reuse the scripts for maintenance, replication, and disaster recovery.
    - You can easily refactor the topology if you desire.

  – Considerations

    - A complete set of administration scripts is a significant investment.
    - You must address script incompatibilities and changes before you migrate.

4. Migrate Web server plug-ins as described in Chapter 6, “Migrating Web server configurations,” on page 97.
5. Set up multiple versions of WebSphere Application Server to coexist.

   No runtime conflicts can exist for multiple instances and versions of WebSphere Application Server to run at the same time on the same machine. Potential conflicts can occur with your port assignments. See “Port number settings in WebSphere Application Server versions” on page 111 for more information.

Overview of migration, coexistence, and interoperability

The goal of migration is to reconstruct your earlier version of WebSphere Application Server in a nearly identical Version 6.1 environment. One goal of coexistence is to create a mixed-version environment that is not in conflict and allows the nodes of all versions to start and run at the same time; another goal is to create an environment that facilitates rollback and allows one or the other version to run at one time. Interoperating is exchanging data between two coexisting product installations or between products on different systems.

WebSphere Application Server Version 6.1 can coexist with Version 5.x and Version 6.0.x. Depending on the previous version of WebSphere Application Server, there might be port conflicts that need to be resolved. See “Coexistence support” on page 105 and Chapter 10, “Configuring ports,” on page 111 for more information.

WebSphere Application Server Version 6.1 migration leverages the existing environment and applications and changes them to be compatible with the WebSphere Application Server Version 6.1 environment. Existing application components and configuration settings are applied to the Version 6.1 environment during the migration process.

If you use an earlier version of WebSphere Application Server, the system administrator might have fine-tuned various application and server settings for your environment. It is important to have a strategy for migrating these settings with maximum efficiency and minimal loss.

You can perform incremental migration of your WebSphere Application Server Version 5.x or 6.0.x configuration by calling the migration tools multiple times, each time specifying a different set of instances or profiles.

Migration involves the following main steps:

1. Testing your applications in a non-production WebSphere Application Server Version 6.1 environment, and making any changes to the applications that are necessary to ensure that they run in that environment.
2. Migrating those applications and your environment to Version 6.1.

   This step can be performed by using the migration tools shipped with the product.

The migration tools migrate applications and configuration information to the new version as described in Chapter 5, “Migrating product configurations,” on page 77. See “Using the migration tools to migrate product configurations” on page 87.

The migration from WebSphere Application Server Version 5.x or 6.0.x to Version 6.1 is fairly routine. Important reference articles for this migration include the following articles:

- Migrating from Version 5 embedded messaging
- Managing WebSphere Version 5 JMS use of WebSphere version 6 messaging resources

If you neither migrate nor coexist with an earlier version of WebSphere Application Server, you are choosing to ignore the previous installation and you can run only one version at a time because of conflicting default port assignments. It is possible for both versions to run at the same time without conflict if you use non-default ports in the earlier version. To set up coexistence with WebSphere Application Server Version 5.x or 6.0.x, ensure that unique ports are selected during profile creation for the Version 6.1 installation. To set up coexistence with an existing installation of Version 6.1, select the radio button that states ″Install a new copy of the V61 Application Server product″ during installation.

You can resolve conflicting port assignments by specifying port assignments for coexistence during profile creation, by **wsadmin** scripting or by using the **Servers > Application Servers > server1 > Ports** administrative console page to ensure that WebSphere Application Server Version 6.1 can run with an earlier version.

Coexistence processing changes the following configuration files:

- virtualhosts.xml
- serverindex.xml

See “Port number settings in WebSphere Application Server versions” on page 111 for more information.

Premigration considerations

Before you begin the process of migrating to WebSphere Application Server Version 6.1, there are some considerations of which you need to be aware.

- Before you migrate to JDK 1.5 (introduced in WebSphere Application Server Version 6.1) from JDK 1.4 (introduced in Version 5.1) or JDK 1.3 (supported in Version 5.0.x) , review your applications for necessary changes based on the Sun Microsystems Java specification.

  See “API and specification migration” on page 74.
- The migration articles assume that WebSphere Application Server Version 6.1 is being installed in an environment where it must coexist with prior levels of WebSphere Application Server.

  Consider the following items while planning to enable coexistence:
  - Update prerequisites to the levels required by WebSphere Application Server Version 6.1.

    Prior levels of WebSphere Application Server will continue to run at the higher prerequisite levels.
  - Review the ports that have been defined to ensure that the WebSphere Application Server Version 6.1 installation does not conflict.

    In particular, note that the default daemon port definition for both versions is the same when installing to coexist with WebSphere Application Server Version 5.x or 6.0.x.

See “Port number settings in WebSphere Application Server versions” on page 111 for default port information.

See Chapter 8, “Coexisting,” on page 105.

- WebSphere Application Server Version 6.1 migration converts HTTP transports to channel-framework Web container transport chains.
- If you create a profile that does not meet the migration requirements (such as naming requirements), you can remove the old profile and create a new one rather than uninstalling and reinstalling the WebSphere Application Server Version 6.1 product.
- The migration tools create a migration backup directory containing a backup copy of the configuration from the previous version. Here are some guidelines that might help you to determine the amount of file-system space that this directory might require.
  - If you are migrating from Version 5.x, the space available for this directory should be at least the size of the previous version’s configuration directory and applications.
  - If you are migrating from Version 6.0.x, the space available for this directory should be at least the size of the previous profile’s configuration directory and applications.
- After you use the migration tools to migrate to WebSphere Application Server Version 6.1, you might need to do some things that are not done automatically by the migration tools.
  - Examine any Lightweight Third Party Authentication (LTPA) security settings that you might have used in WebSphere Application Server Version 5.x or 6.0.x, and make sure that Version 6.1 security is set appropriately.
  - Check the WASPostUpgrade.log file in the logs directory for details about any JSP objects that the migration tools did not migrate.

    If Version 6.1 does not support a level for which JSP objects are configured, the migration tools recognize the objects in the output and log them.
  - Verify the results of the automatic Cloudscape database migration, and manually migrate any Cloudscape databases that are not automatically migrated by the tools.

    See “Migrating Cloudscape databases” on page 95.

API and specification migration

Migrating application programming interfaces (APIs) and specifications involves moving to the current Java component level as well as to other technologies that WebSphere Application Server Version 6.1 supports.

If your existing applications currently support different specification levels than are supported by this version of the product, it is likely that you must update at least some aspects of the applications to comply with the new specifications.

In many cases, IBM provides additional features and customization options that extend the specification level even further. If your existing applications use IBM extensions from earlier product versions, it might be necessary for you to perform mandatory or optional migration to use the same kinds of extensions in Version 6.1.

The following table summarizes potential migration areas.

Functional area	Support in Version 5.0.x	Support in Version 5.1	Support in Version 6.0.x	Support in Version 6.1	Migration details
Enterprise JavaBeans (EJB)	EJB 2.0	EJB 2.0	EJB 2.1	EJB 2.1	Full support for EJB 2.0 and 2.1 is provided.
Java Connector Architecture (JCA)	JCA 1.0	JCA 1.0	JCA 1.5	JCA 1.5	Java 2 Connector support was completed in Version 6.0. Some changes might be necessary to take full advantage of this support.

Functional area	Support in Version 5.0.x	Support in Version 5.1	Support in Version 6.0.x	Support in Version 6.1	Migration details
Java Database Connectivity (JDBC) API	JDBC 2.0	JDBC 2.0	JDBC 3.0	JDBC 3.0	Many applications can run unchanged in Version 6.1 although some changes might be required or recommended.
JavaServer Pages (JSP)	JSP 1.2	JSP 1.2	JSP 2.0	JSP 2.0	For more information, see the Web applications article in the information center.
Security	IBM security	IBM security	IBM security	IBM security	Changes might be required due to J2EE security.
Java Servlets	Servlet 2.3	Servlet 2.3	Servlet 2.4	Servlet 2.4	Many Servlet 2.3 and 2.4 applications can run unchanged in Version 6.1 although changes might be required or recommended.
Sessions	IBM sessions	IBM sessions	IBM sessions	IBM sessions	Many applications can run unchanged in Version 6.1 although changes might be required or recommended.
Transactions	IBM transactions	IBM transactions	IBM transactions	IBM transactions	There was a change in the import statement in Version 6.0.x. Also, one datasource connection cannot be used across multiple user transactions.
Web services	Apache SOAP 2.3 in Version 5.0 and 5.0.1; Web Services for J2EE 1.1 in Version 5.0.2	Web Services for J2EE 1.1	Web Services for J2EE 1.1	Web Services for J2EE 1.1	The Apache SOAP Web services support provided in WebSphere Application Server Version 5.0 and 5.0.1 was deprecated in Version 5.0.2 and is not supported in Version 6.1. Applications that are using this SOAP implementation should migrate to Web Services for J2EE.
XML parser	XML4J 2.0.x	XML4J 4.0.6	XML4J 4.0.6	XML4J 4.2.2	Changes to move to the supported API XML4J Version 4.2.2 level are required.
					Recompilation is required to migrate a Version 5.x XML application to the Version 6.x level.
XML transformer	XSLT4J 2.5.4	XSLT4J 2.5.4	XSLT4J 2.5.4	XSLT4J 2.5.4	Changes are required to move to the supported XSLT4J 2.5.4 transformer levels.

Notes on the use of JDK 1.5:

- WebSphere Application Server began supporting JDK 1.5 in Version 6.1.
- In general, existing Version 5.x and 6.0.x application binaries that were developed using JDK 1.3 or 1.4 are highly compatible and usually do not require modifications to run. However, recompilation of the JDK 1.3 or 1.4 applications at the JDK 1.5 level might necessitate modifications of the source code to conform to incompatible changes present in JDK 1.5. As part of your migration planning, you should review the JDK compatibility restrictions that are documented by Sun Microsystems at Java 2 Platform Standard Edition 5.0 Compatibility with Previous Releases.
- A mixed cell containing Version 5.x or 6.0.x and Version 6.1 nodes requires that all application binaries remain at the lowest JDK level used. Although you can successfully migrate Version 5.x or 6.0.x applications to Version 6.1, this is only meant to be a temporary state as you transition to Version 6.1. After you begin migration to Version 6.1, you should plan to complete the migration of the entire cell, update your tooling to Version 6.1, and update your applications to conform to JDK 1.5 requirements. This should be done before any further application changes. After you have completely migrated your cell to Version 6.1, you should upgrade your application binaries to the JDK

1.5 level the next time you make application modifications that require recompiling. This might require source code changes to your application to conform to the JDK 1.5 API changes as documented by Sun Microsystems.

Chapter 5. Migrating product configurations

Use the migration tools to migrate WebSphere Application Server product configurations.

See “Overview of migration, coexistence, and interoperability” on page 72 and “Premigration considerations” on page 73.

The following configuration upgrades of WebSphere Application Server versions and offerings are directly supported.

Table 2. Directly Supported Configuration Upgrades

Migration Source	WebSphere Application Server Version 6.1 Target
	Base Stand-alone Profile
WebSphere Application Server Version 5.x or 6.0.x base stand-alone application server	Supported
WebSphere Application Server Version 5.x or 6.0.x Network Deployment stand-alone application server	Supported
WebSphere Application Server Version 5.x or 6.0.x Network Deployment federated application server	Supported
WebSphere Application Server Version 5.x or 6.0.x Network Deployment deployment manager	
WebSphere Application Server Version 5.x or 6.0.x Express stand-alone application server	Supported

You can migrate your product configurations using the WebSphere Application Server Version 6.1 command-line migration tools. These migration tools support migration from WebSphere Application Server Version 5.x and Version 6.0.x.

Before using the migration tools, consult the WebSphere Application Server Version 6.1 Release Notes document to understand what fixes you must apply to earlier versions. Applying fixes to an earlier version might also apply fixes to files that have a role in the migration. Apply any fixes to ensure the most effective migration of configurations and applications possible.

When you use the migration tools, the overall migration process includes these steps:

1. Install the WebSphere Application Server Version 6.1 product.

   Installing the product creates a stand-alone application server.
2. Use the migration tools to migrate the previous release to the WebSphere Application Server Version 6.1 product.

   See “Using the migration tools to migrate product configurations” on page 87.

For help in troubleshooting problems when migrating, see Chapter 11, “Troubleshooting migration,” on page 115.

1. Migrate the WebSphere Application Server Version 5.x and Version 6.0.x stand-alone application server nodes.

   Select one of the following migration scenarios for information about how to migrate configuration data to a Version 6.1 stand-alone application server:

   - “Migrating to a Version 6.1 stand-alone application server profile” on page 83
   - “Migrating the Version 5.x or 6.0.x default instance to the default Version 6.1 stand-alone application server profile” on page 85

2. You might want to do some things that are not done automatically by the migration tools.
   - Perform any of the following non-automated tasks:
     – "Migrating to Version 3 of the UDDI registry" on page 39
     – Migrating from Version 5 embedded messaging
   - Examine any Lightweight Third Party Authentication (LTPA) security settings that you might have used in WebSphere Application Server Version 5.x or 6.0.x, and make sure that Version 6.1 security is set appropriately.
   - Check the WASPostUpgrade.log file in the logs directory for details about any JSP objects that the migration tools did not migrate.

     If Version 6.1 does not support a level for which JSP objects are configured, the migration tools recognize the objects in the output and log them.
   - Configure WebSphere Application Server to use a database.

     For example, you can configure WebSphere Application Server to use DB2.
3. Use "Configuration mapping during product-configuration migration" to verify the results of the migration.

   The article has a detailed description of how the migration tools migrate objects and what you should verify.

Configuration mapping during product-configuration migration

Various configurations are mapped during product-configuration migration.

Migration always involves migrating a single profile to another single profile on the same machine or a separate machine. Go to the information center for the WebSphere Application Server Network Deployment product to learn how the migration tools map models, clones, server groups, clusters, and Lightweight Third Party Authentication (LTPA) security settings.

Many migration scenarios are possible. The migration tools map objects and attributes existing in the version from which you are migrating to the corresponding objects and attributes in the Version 6.1 environment.

Bootstrap port

The migration tools map a non-default value directly into the Version 6.1 environment.

If the -portBlock parameter is specified during the call to **WASPostUpgrade**, however, a new port value is given to each application server that is migrated to Version 6.1.

Command-line parameters

The migration tools convert appropriate command-line parameters to Java virtual machine (JVM) settings in the server process definition. Most settings are mapped directly. Some settings are not migrated because their roles in the WebSphere Application Server Version 6.1 configuration do not exist, have different meanings, or have different scopes.

Generic server

In WebSphere Application Server Version 5.1.x, a generic server was an APPLICATION_SERVER fitted to manage external resources. In Version 6.0.x and later, it has its own type called GENERIC_SERVER. Migration will perform this conversion, but migration cannot accurately migrate the external resources that the generic server references. After migration has completed migrating the generic server settings, you might need to perform additional tasks. If the old resource that the generic server was managing is located under the old WebSphere Application Server installation, perform the following tasks:

1. Copy any related files to the new installation.
2. Run any setup required to put the external application back into a valid and working state.

It is best that you reinstall the resource into the new WebSphere Application Server directory. Whatever you choose to do, the final step is to reset the reference to the new location of the application.

If the old resource that the generic server was managing is not installed under the old WebSphere Application Server installation, nothing further is required.

Policy file

WebSphere Application Server Version 6.1 migrates all the policy files that are installed with Version 5.x or 6.0.x by merging settings into the Version 6.1 policy files with the following characteristics:

- Any comments located in the Version 6.1 policy file will be preserved. Any comments contained in the Version 5.x or 6.0.x policy file will not be included in the Version 6.1 file.
- Migration will not attempt to merge permissions or grants; it is strictly an add-type migration. If the permission or grant is not located in the Version 6.1 file, the migration will bring it over.
- Security is a critical component; thus, the migration makes any additions at the end of the original .policy file right after the comment `MIGR0372I: Migrated grant permissions follow`. This is done to help administrators verify any policy file changes that the migration has made.

Properties, classes, and lib/app directories

Migration copies files from prior version directories into the WebSphere Application Server Version 6.1 configuration.

Property files

WebSphere Application Server Version 6.1 migrates all the property files that are installed with Version 5.x or 6.0.x by merging settings into the Version 6.1 property files with these exceptions for Version 5.x files:

- j2c.properties (migrated into resources.xml files)
- samples.properties

Migration does not overlay property files.

Resource adapter archives (RARs) referenced by J2C resources

RARs that are referenced by J2C resources are migrated if those RARs are in the old WebSphere Application Server installation. In this case, the RARs are copied over to the corresponding location in the new WebSphere Application Server installation. Relational Resource Adapter RARs will not be migrated.

Samples

No migration of samples from previous versions is available. There are equivalent WebSphere Application Server Version 6.1 samples that you can install.

Security

Java 2 security is enabled by default when you enable security in WebSphere Application Server Version 6.1. Java 2 security requires you to grant security permissions explicitly.

There are several techniques that you can use to define different levels of Java 2 security in Version 6.1. One is to create a was.policy file as part of the application to enable all security permissions. The migration tools call the **wsadmin** command to add an existing was.policy file in the Version 6.1 properties directory to enterprise applications as they are being migrated.

When migrating to WebSphere Application Server Version 6.1, your choice of whether or not to migrate to support script compatibility results in one of two different outcomes.

- If you choose to migrate to support script compatibility, your security configuration is brought over to Version 6.1 without any changes.

This is the default.

- If you choose not to migrate to support script compatibility, the security configuration is converted to the default configuration for WebSphere Application Server Version 6.1. The default security configuration for Version 6.1 acts almost the same as in the previous versions, but there are some changes.

  For example, existing keyfiles and trustfiles are moved out of the SSLConfig repertoire and new keystore and truststore objects are created.

For more information on migrating your security configurations to Version 6.1, see "Migrating, coexisting, and interoperating – Security considerations" on page 50.

Stdin, stdout, stderr, passivation, and working directories

The location for these directories is typically the logs directory under the WebSphere Application Server profile directory. For WebSphere Application Server Version 6.1, the default location for the stdin, stdout, and stderr files is the logs directory located under the WebSphere Application Server profile directory—for example, the logs directory for the default profile is /QIBM/UserData/WebSphere/AppServer/V61/Base/profiles/default/logs.

The migration tools attempt to migrate existing passivation and working directories. Otherwise, appropriate Version 6.1 defaults are used.

In a coexistence scenario, using common directories between versions can create problems.

Transport ports

The migration tools migrate all ports. The tools log a port-conflict warning if a port is already defined in the configuration. You must resolve any port conflicts before you can run servers at the same time.

If the -portBlock parameter is specified in the **WASPostUpgrade** command, a new value is assigned to each transport that is migrated. Choosing -scriptCompatibility=″true″ or -scriptCompatibility=″false″ results in two different outcomes for transport ports if you are migrating from WebSphere Application Server Version 5.x:

- -scriptCompatibility=″true″

  This results in your transport ports being brought over as they are. This is the default.

- -scriptCompatibility=″false″

  This results in the transport ports being converted to the implementation of channels and chains. From an external application usage standpoint, they will still act the same; but they have been moved to the TransportChannelService.

For more information on the **WASPostUpgrade** command, see "WASPostUpgrade command" on page 91.

Web modules

The specification level of the Java 2 Platform, Enterprise Edition (J2EE) implemented in WebSphere Application Server Version 6.0.x required behavior changes in the Web container for setting the content type. If a default servlet writer does not set the content type, not only does the Web container no longer default to it but the Web container returns the call as ″null.″ This situation might cause some browsers to display resulting Web container tags incorrectly. To prevent this problem from occurring, migration sets the autoResponseEncoding IBM extension to ″true″ for Web modules as it migrates enterprise applications.

Preparing for product-configuration migration

You must prepare your system for migrating to WebSphere Application Server Version 6.1.

See “Overview of migration, coexistence, and interoperability” on page 72 and “Premigration considerations” on page 73.

For help in troubleshooting problems when migrating, see Chapter 11, “Troubleshooting migration,” on page 115.

1. Install WebSphere Application Server Version 6.1.
2. Familiarize yourself with the tools and features of WebSphere Application Server Version 6.1.
3. Verify that you have the minimum prerequisites required for migration.
   For more information, see “Product-configuration migration prerequisites.”
4. Evaluate the items deprecated in WebSphere Application Server Version 6.1.
   For more information, see “Deprecated and removed features” on page 1.
5. Evaluate the changes to API specification levels to determine which applications you need to migrate.
   To plan your application migration requirements, use “API and specification migration” on page 74.
6. Evaluate the changes to configuration settings.
   For more information, see “Configuration mapping during product-configuration migration” on page 78.

Product-configuration migration prerequisites

Before you migrate your old version of WebSphere Application Server to Version 6.1, you must meet certain requirements.

- Make sure that you have the minimum version of source WebSphere Application Server that is required.
  - If you are migrating from WebSphere Application Server 5.0.x, you must be at Version 5.0 or higher.
  - If you are migrating from WebSphere Application Server Version 5.1.x, you must be at Version 5.1 or higher.
  - If you are migrating from WebSphere Application Server 6.0.x, you must be at Version 6.0 or higher.

  To determine the current level of WebSphere Application Server installed on your system, perform these steps:

  Version 5.0.x:

  1. Enter the following command on the command line:

     `WRKLNK '/QIBM/ProdData/`*`product_dir`*`/properties/`*`version`*`/`*`product_file`*

     where *product_dir* is:
     - `WebASE/ASE5` for Express Version 5.0.x
     - `WebAS5/Base` for base Version 5.0.x
     - `WebAS5/ND` for Network Deployment Version 5.0.x

     and *product_file* is:
     - `Express.product` for Express Version 5.0.x
     - `BASE.product` for base Version 5.0.x
     - `ND.product` for Network Deployment Version 5.0.x
  2. Specify option 5 (Display) next to the product file to view the contents. The number within the <version> tags show the current version you have installed.

  Version 5.1.x:

  1. Enter the following command on the command line:

     `WRKLNK '/QIBM/ProdData/`*`product_dir`*`/properties/`*`version`*`/`*`product_file`*

     where *product_dir* is:
     - `WebASE51/ASE` for Express Version 5.1.x
     - `WebAS51/Base` for base Version 5.1.x

– `WebAS51/ND` for Network Deployment Version 5.1.x

and *product_file* is:

– `Express.product` for Express Version 5.1.x

– `BASE.product` for base Version 5.1.x

– `ND.product` for Network Deployment Version 5.1.x

2. Specify option 5 (Display) next to the product file to view the contents. The number within the <version> tags shows the current version that you have installed.

Version 6.0.x:

1. Enter the following command on the command line:

   `WRKLNK '/QIBM/ProdData/WebSphere/AppServer/V6/`*product_dir*`/properties/`*version*`/WAS.product`

   where *product_dir* is:

   – `Base` for base and Express Version 6.0.x

   – `ND` for Network Deployment Version 6.0.x

2. Specify option 5 (Display) next to the product file to view the contents. The number within the <version> tags shows the current version that you have installed.

If you do not meet the minimum version, obtain the latest group PTF. See WebSphere Application Server PTFs for iSeries for information on the correct group PTF for your OS/400 or i5/OS release level and WebSphere Application Server Version 5.x or 6.0.x product.

- Make sure that WebSphere Application Server Version 6.1 is installed.

  WebSphere Application Server Version 6.1 Network Deployment needs to be installed if you are migrating to Network Deployment.

- Make sure that your user profile has *ALLOBJ authority.

  When you call the **WASPreUpgrade** and **WASPostUpgrade** migration tools, your user profile must have *ALLOBJ authority.

Migrating profiles

After you have migrated your applications, you need to migrate your instance configurations to profiles.

See “Overview of migration, coexistence, and interoperability” on page 72 and “Premigration considerations” on page 73.

See “Product-configuration migration prerequisites” on page 81 to see how to determine the currently installed product level of WebSphere Application Server.

Select the appropriate option to obtain instructions on how to migrate from your old version of WebSphere Application Server to a new or default Version 6.1 profile.

- “Migrating to a Version 6.1 stand-alone application server profile” on page 83

  This article contains instructions for migrating a WebSphere Application Server Version 5.x or 6.0.x profile to a Version 6.1 stand-alone application server profile.

- “Migrating the Version 5.x or 6.0.x default instance to the default Version 6.1 stand-alone application server profile” on page 85

  This article contains instructions for migrating the WebSphere Application Server Version 5.x or 6.0.x default instance to a Version 6.1 stand-alone application server profile.

Tip: For help in troubleshooting problems when migrating, see Chapter 11, “Troubleshooting migration,” on page 115.

Migrating to a Version 6.1 stand-alone application server profile

Use the migration tools to migrate from WebSphere Application Server Version 5.x or 6.0.x to a new Version 6.1 stand-alone application server profile.

See “Overview of migration, coexistence, and interoperability” on page 72 and “Premigration considerations” on page 73.

For help in troubleshooting problems when migrating, see Chapter 11, “Troubleshooting migration,” on page 115.

Before following these instructions, perform the actions in “Preparing for product-configuration migration” on page 80.

Tip: Before migrating a WebSphere Application Server Version 5.x or 6.0.x stand-alone application server profile, use the **backupConfig** command or your own preferred backup utility to back up your existing configuration if you want to be able to restore it to its previous state after migration. Make sure that you note the exact name and location of this backed-up configuration.

1. Create a WebSphere Application Server Version 6.1 profile to receive the Version 5.x or 6.0.x configuration.
   a. Start the Qshell environment so that you can run WebSphere Application Server scripts.

      Enter the following command on a command line:

      ```
      STRQSH
      ```

   b. Run the **dspwasinst** script to obtain the node name and server name for the Version 5.x or 6.0.x instance or profile that is to be migrated.

      Use the following parameters:

      ```
      app_server_root/bin/dspwasinst
       -instance 5.x_or_6.0.x_profile_name
      ```

      where

      - *app_server_root* is the location of the Version 5.x or 6.0.x installation that contains the instance or profile to be migrated
      - *5.x_or_6.0.x_profile_name* is the name of the Version 5.x or 6.0.x instance or profile that is to be migrated

      The name of the Version 5.x or 6.0.x node is listed in the **Node** section, and the name of the server is listed in the **Information for server** section.

   c. Run the **manageprofiles** script.

      Use the following parameters:

      ```
      app_server_root/bin/manageprofiles
       -create
       -profileName 6.1_profile_name
       -startingPort starting_port_number
       -templatePath app_server_root/profileTemplates/default
       -serverName 5.x_or_6.0.x_application_server_name
       -nodeName 5.x_or_6.0.x_node_name
      ```

      where

      - *app_server_root* is the location where Version 6.1 is installed
      - *6.1_profile_name* is the name of your Version 6.1 profile

        This parameter must be identical to the Version 5.x or 6.0.x instance or profile that is to be migrated. The source and target node names must be identical when migrating to Version 6.1.
      - *starting_port_number* is the first of a block of 13 consecutive ports

- *5.x_or_6.0.x_application_server_name* is the name of the Version 5.x or 6.0.x application server obtained in the previous step
- *5.x_or_6.0.x_node_name* is the Version 5.x or 6.0.x node name obtained in the previous step

For details on the syntax and parameters of the **manageprofiles** command, see the information center.

2. Save the WebSphere Application Server Version 5.x or 6.0.x configuration.
   a. Start the Qshell environment so that you can run WebSphere Application Server scripts.

      Enter the following command on a command line:

      ```
      STRQSH
      ```

   b. Run the **WASPreUpgrade** script.

      Use the following parameters:

      ```
      app_server_root/bin/WASPreUpgrade
       backup_directory_name
       profile_root
      ```

      where

      - *app_server_root* is the location where Version 6.1 is installed
      - *backup_directory_name* (required parameter) is the fully qualified path to the integrated file system directory where the **WASPreUpgrade** migration tool stores the saved configuration and files

        The directory is created if it does not already exist. It is also the directory where the **WASPreUpgrade** migration tool writes a log file called WASPreUpgrade.log that chronicles the steps taken by the **WASPreUpgrade** command.
      - *profile_root* (required parameter) is the path to the Version 5.x or 6.0.x instance or profile that is to be migrated

      For a full explanation of the **WASPreUpgrade** command and its parameters, see “WASPreUpgrade command” on page 90.
3. Restore the WebSphere Application Server Version 5.x or 6.0.x configuration into a Version 6.1 profile.
   a. Start the Qshell environment so that you can run WebSphere Application Server scripts.

      Enter the following command on a command line:

      ```
      STRQSH
      ```

   b. Run the **WASPostUpgrade** script.

      Use the following parameters:

      ```
      app_server_root/bin/WASPostUpgrade
       backup_directory_name
       -profileName 6.1_profile_name
       [-portBlock port_starting_number]
      ```

      where

      - *app_server_root* is the location where Version 6.1 is installed
      - *backup_directory_name* is the required name of the directory in which the **WASPreUpgrade** tool stored the saved configuration and files and from which the **WASPostUpgrade** tool reads the configuration and files
      - *6.1_profile_name* is the name of the Version 6.1 profile to which the script migrates your configuration
      - *port_starting_number* specifies the first of a block of 10-15 consecutive port numbers that are not in use on the iSeries server where the migration is being performed

        It is recommended that you always specify the -portBlock parameter if you do not want your profile’s ports to conflict with the default profile’s ports.

For a full explanation of the **WASPostUpgrade** command and its parameters, see "WASPostUpgrade command" on page 91.

4. Start the WebSphere Application Server Version 6.1 profile that receives the Version 5.x or 6.0.x configuration.
   a. Start the QWAS61 subsystem if it is not already started.

      Enter the following command on a command line:

      ```
      STRSBS QWAS61/QWAS61
      ```

   b. Start the Qshell environment so that you can run WebSphere Application Server scripts.

      Enter the following command on a command line:

      ```
      STRQSH
      ```

   c. Run the **startServer** script.

      Use the following parameters:

      ```
      app_server_root/bin/startServer
       -profileName 6.1_profile_name
       5.x_or_6.0.x_server_name
      ```

      where

      - *app_server_root* is the location where Version 6.1 is installed
      - *6.1_profile_name* is the name of the Version 6.1 profile created in an earlier step
      - *5.x_or_6.0.x_server_name* is the name of the Version 5.x or 6.0.x application server that was migrated

Migrating the Version 5.x or 6.0.x default instance to the default Version 6.1 stand-alone application server profile

Use the migration tools to migrate from a Version 5.x or 6.0.x default instance to the default Version 6.1 stand-alone application server profile.

See "Overview of migration, coexistence, and interoperability" on page 72 and "Premigration considerations" on page 73.

For help in troubleshooting problems when migrating, see Chapter 11, "Troubleshooting migration," on page 115.

Before following these instructions, take the steps in "Preparing for product-configuration migration" on page 80.

1. Save the WebSphere Application Server Version 5.x or 6.0.x configuration.
   a. Start the Qshell environment so that you can run WebSphere Application Server scripts.

      Enter the following command on a command line:

      ```
      STRQSH
      ```

   b. Run the **WASPreUpgrade** script.

      Use the following parameters when migrating to a Version 6.1 stand-alone application server profile:

      ```
      app_server_root/bin/WASPreUpgrade
       backup_directory_name
       old_default_instance
      ```

      where

      - *app_server_root* is the location where Version 6.1 is installed
      - *backup_directory_name* (required parameter) is the fully qualified path to the directory where the **WASPreUpgrade** migration tool stores the saved configuration and files

This directory is created if it does not already exist. This directory also stores a log file called WASPreUpgrade.log that chronicles the steps taken by the **WASPreUpgrade** command.

- *old_default_instance* is one of the following, depending on the version from which you are migrating:
  - /QIBM/UserData/WebAS5/Base/default
  - /QIBM/UserData/WebAS51/Base/default
  - /QIBM/UserData/WebSphere/AppServer/V6/Base/profiles/default

The **WASPreUpgrade** tool writes a log file called WASPreUpgrade.log that chronicles the steps taken by the **WASPreUpgrade** command.

For a full explanation of the **WASPreUpgrade** command and its parameters, see "WASPreUpgrade command" on page 90.

2. Restore the WebSphere Application Server Version 5.x or 6.0.x configuration into a Version 6.1 profile.
   a. Start the Qshell environment so that you can run WebSphere Application Server scripts.

      Enter the following command on a command line:

      ```
      STRQSH
      ```

   b. Run the **WASPostUpgrade** script.

      Use the following parameters:

      ```
      app_server_root/bin/WASPostUpgrade
      backup_directory_name
      -profileName default
      ```

      where

      - *app_server_root* is the location where Version 6.1 is installed
      - *backup_directory_name* (required parameter) is the fully qualified path to the directory where the **WASPreUpgrade** migration tool previously saved the WebSphere Application Server Version 5.x or 6.0.x instance configuration

      For a full explanation of the **WASPostUpgrade** command and its parameters, see "WASPostUpgrade command" on page 91.

3. Start the WebSphere Application Server Version 6.1 default profile.
   a. Start the Qshell environment so that you can run WebSphere Application Server scripts.

      Enter the following command on a command line:

      ```
      STRQSH
      ```

   b. If the QWAS61 subsystem has not been started, start the default profile.

      Enter the following command on a command line:

      ```
      STRSBS QWAS61/QWAS61
      ```

   c. If the QWAS61 subsystem is already started and the SERVER1 job is present, stop the server and start it again.
   d. If the QWAS61 subsystem is already started but the SERVER1 job is not present, start the server using the **startServer** script.

      Enter the following commands on a command line:

      ```
      STRQSH

      app_server_root/bin/startServer
      ```

      where *app_server_root* is the location where Version 6.1 is installed.

Using the migration tools to migrate product configurations

Migration support consists of tools that are shipped as part of WebSphere Application Server Version 6.1 that primarily provide support for saving the configuration and applications from an older version of WebSphere into a migration-specific backup directory and then importing that configuration into Version 6.1.

See “Overview of migration, coexistence, and interoperability” on page 72 and “Premigration considerations” on page 73.

All of the migration scripts are in the *app_server_root*/bin directory after installation. The **WASPreUpgrade** tool also ships on the utilities CD-ROM so that you can store the configuration of an existing release before installing the Version 6.1 product. It is important to use the migration tools for the version of WebSphere Application Server that you are installing. The tools change over time. The tools on the product CD-ROM provide the necessary function for migrating from a previous release of WebSphere Application Server to the one on the product CD-ROM. The tools on the CD-ROM match the product on the CD-ROM. If you use migration tools from an earlier release of WebSphere Application Server, you are likely to encounter a problem with the migration.

Select the appropriate migration tools to migrate your product configurations.

clientUpgrade
: Upgrades the client application to a new release level.

convertScriptCompatibility
: Used by administrators to convert their configuration from a mode that supports backward compatibility of Version 5.x or 6.0.x administration scripts to a mode that is fully Version 6.1.

  See “convertScriptCompatibility command” on page 88.

WASPreUpgrade
: Saves the applications and configuration data from a previous installation of WebSphere Application Server to a backup directory.

  The **WASPostUpgrade** script restores the configuration data from the directory to the new installation.

  See “WASPreUpgrade command” on page 90.

WASPostUpgrade
: Restores the configuration data from a previous release.

  WASPostUpgrade reads the data from the backup directory where the **WASPreUpgrade** script stored the data.

  See “WASPostUpgrade command” on page 91.

Tip: For help in troubleshooting problems when migrating, see Chapter 11, “Troubleshooting migration,” on page 115.

Use the selected tools to migrate your product configuration.

clientUpgrade script

The clientUpgrade script migrates application client modules and their resources in an enterprise archive (EAR) file so that these application clients can run in WebSphere Application Server Version 6.x. The script converts an EAR file that you want to migrate and then overwrites the original EAR file with the converted EAR file.

The following title provides information about the clientUpgrade script.

Note: This command is deprecated in Version 6.1.

Type	Description
Product	The clientUpgrade script is available in the WebSphere Application Server (Express and Base) product only.
Authority	To run this script, your user profile must have *ALLOBJ authority.

Syntax	The syntax of the clientUpgrade script is:
	`clientUpgrade` *`EAR_file`* `[ -clientJAR` *`client_JAR_file`* `]`
	`[ -logFileLocation` *`logFileLocation`* `]`
	`[ -traceString` *`trace_spec`* `[ -traceFile` *`file_name`* `] ]`

Parameters	The parameters of the clientUpgrade script are: • *EAR_file* -- This is a required parameter. The value EAR_file specifies the fully-qualified path of the EAR file that contains the application client modules that you want to migrate. • -clientJAR -- This is an optional parameter. The value client_JAR_file specifies a JAR file that you want to migrate. The script overwrites the original EAR file with a new EAR file that contains only the specified JAR files. If you do not specify this parameter, the clientUpgrade script migrates all client JAR files in the EAR file. • -logFileLocation -- Use this optional parameter to specify an alternate location to store the log output. • -traceString -- This is an optional parameter. The value trace_spec specifies the trace information that you want to collect. To gather all trace information, specify ″*=all=enabled″ (including the double quotation marks (″)). By default, the script does not gather trace information. If you specify this parameter, you must also specify the -traceFile parameter. • -traceFile -- This is an optional parameter. The value file_name The value file_name specifies the name of the output file for trace information. If you specify the -traceString parameter but do not specify the -traceFile parameter, the script does not generate a trace file.
Logging	The clientUpgrade script displays status while it runs. It also saves more extensive logging information to the clientupgrade.log file. This file is located in the /QIBM/UserData/WebSphere/AppServer/V6/edition/profiles/default/logs directory (for a default installation using the default profile) or in the location specified by the -logFileLocation parameter.

These examples demonstrate correct syntax. In this example, the My50Application.ear file is migrated from WebSphere Application Server Version 5.0.x, The script overwrites the original EAR file with a new file that you can deploy in your WebSphere Application Server Version 6.1 profile.

```
clientUpgrade /My50Application/My50Application.ear
```

In this example, only the `myJarFile.jar` client JAR file is migrated. The script overwrites `My50Application.ear` with an EAR file that contains `myJarFile.jar`. You can deploy the new EAR file in your WebSphere Application Server profile.

```
clientUpgrade /My50Application/My50Application.ear -clientJAR myJarFile.jar
```

convertScriptCompatibility command

The **convertScriptCompatibility** command is used by administrators to convert their configurations from a mode that supports backward compatibility of WebSphere Application Server Version 5.x or 6.0.x administration scripts to a mode that is fully in the Version 6.1 configuration model.

The scope of the configuration changes depend on the type of profile that is being processed.

- For Express and base configurations, the default is to convert all servers owned by the node in that configuration.

  Use the -serverName parameter for more granular control.
- For Network Deployment configurations, the default behavior is to convert all nodes and all servers owned by those nodes.

  Use the -nodeName and -serverName parameters for more granular control.

Nodes are checked to verify that they are at a WebSphere Application Server Version 6.1 level before they are processed in order to support mixed-node configurations. Client environments are not processed.

The following conversions take place with this tool:

- processDef to processDefs

  WCCM objects of type processDef from WebSphere Application Server Version 5.x are converted to use processDefs as defined in the Version 6.1 server.xml model. The existing processDef object remains in the configuration and is ignored by the runtime.
- transports to channels

  Existing transport entries in the configuration from WebSphere Application Server Version 5.x are mapped to channel support. This affects server.xml and serverindex.xml files. The values of the transport settings are used to create new channel entries.
- SSL configuration

  WebSphere Application Server Version 6.1 contains enhancements to SSL configuration that result in refactoring the existing SSL configuration model. Both the old and the new model are supported. The default is to map to the WebSphere Application Server Version 5.x or 6.0.x SSL configuration model.

Syntax

The syntax is as follows:

```
convertScriptCompatibility [-help]

convertScriptCompatibility [-profileName profile_name]
                     [-backupConfig true | false]
                     [-nodeName node_name  [-serverName server_name]]
                     [-traceString trace_spec  [-traceFile file_name]]
```

Parameters

Supported arguments include the following parameters:

-help
This displays help for this command

-backupConfig
This is an optional parameter that is used to back up the existing configuration of the current profile. The default is true—that is, to use the **backupConfig** command to save a copy of the current configuration into the *profile_name*/temp directory.

Use the **restoreConfig** command to restore that configuration as required.

-profileName
This is an optional parameter that is used to specify the profile configuration in the Version 6.1 environment. If this is not specified, the default profile is used. If the default profile has not been set or cannot be found, the system returns an error.

-nodeName
This is an optional parameter that is used to specify a particular node name be processed rather than every node in the configuration. If this is not specified, all nodes in the configuration are converted.

-serverName
This is an optional parameter that is used to specify a particular server name to be processed rather than every server in the configuration. It can be used on all profile types and can be used in conjunction with the -nodeName parameter when processing Network Deployment configurations. If this parameter is not specified, all servers in the configuration are converted. If it is used in conjunction with the -nodeName parameter, all processing is limited to the specified node name.

-traceString
This is an optional parameter. The value *trace_spec* specifies the trace information that you want to collect. To gather all trace information, specify `"*=all=enabled"` (with quotation marks). The default is to not gather trace information. If you specify this parameter, you must also specify the -traceFile parameter.

-traceFile
This is an optional parameter. The value *file_name* specifies the name of the output file for trace information. If you specify the -traceString parameter but do not specify the -traceFile parameter, the command does not generate a trace file.

WASPreUpgrade command

The **WASPreUpgrade** command for WebSphere Application Server Version 6.1 saves the configuration of a previously installed version of WebSphere Application Server into a migration-specific backup directory.

Authority

To run this command script, your user profile must have *ALLOBJ authority.

Syntax

The script syntax is as follows:

```
WASPreUpgrade backupDirectory
              currentWebSphereDirectory
              [-traceString trace_spec [-traceFile file_name ]]
```

Parameters

The parameters are as follows:

backupDirectory
This is a required parameter and must be the first parameter that you specify. The value *backupDirectory* specifies the name of the directory where the command script stores the saved configuration.

This is also the directory from which the **WASPostUpgrade** command reads the configuration.

If the directory does not exist, the **WASPreUpgrade** command script creates it.

currentWebSphereDirectory
This is a required parameter and must be the second parameter that you specify.

The value *currentWebSphereDirectory* specifies the name of the instance or profile root directory for the current WebSphere Application Server Version 5.x instance or 6.0.x profile that you want to migrate.

- **For Version 5.0.x Express:** /QIBM/UserData/WebASE/ASE5/*instance*
- **For Version 5.0.x base:** /QIBM/UserData/WebAS5/Base/*instance*
- **For Version 5.0.x Network Deployment:** /QIBM/UserData/WebAS5/ND/*instance*
- **For Version 5.1.x Express:** /QIBM/UserData/WebASE51/ASE/*instance*
- **For Version 5.1.x base:** /QIBM/UserData/WebAS51/Base/*instance*
- **For Version 5.1.x Network Deployment:** /QIBM/UserData/WebAS51/ND/*instance*

- In Version 6.0.x, the profile root may be a unique value chosen during profile creation but the following directories are the defaults:
  - **For Version 6.0.x Express or base:** /QIBM/UserData/WebSphere/AppServer/V6/Base/profiles/*profile*
  - **For Version 6.0.x Network Deployment:** /QIBM/UserData/WebSphere/AppServer/V6/ND/profiles/*profile*

-traceString
This is an optional parameter. The value *trace_spec* specifies the trace information that you want to collect.

To gather all trace information, specify `"*=all=enabled"` (with quotation marks).

If you do not specify the -traceString or -traceFile parameter, the command creates a trace file by default and places it in the *backupDirectory*/logs directory.

If you specify this parameter, you must also specify the -traceFile parameter.

-traceFile
This is an optional parameter. The value *file_name* specifies the name of the output file for trace information.

If you do not specify the -traceString or -traceFile parameter, the command creates a trace file by default and places it in the *backupDirectory*/logs directory.

If you specify the -traceString parameter but do not specify the -traceFile parameter, the script does not generate a trace file.

Logging

The **WASPreUpgrade** tool displays status to the screen while it runs. The tool also saves a more extensive set of logging information in the WASPreUpgrade.*time_stamp*.log file written to the *backupDirectory* directory, where *backupDirectory* is the value specified for the backupDirectory parameter. You can view the WASPreUpgrade.*time_stamp*.log file with a text editor.

Migrated resources

WASPreUpgrade saves all of your resources, but it does not migrate entities in your classes directory.

Migration saves the following files in the *backupDirectory* directory.
- config
- properties

WASPostUpgrade command

The **WASPostUpgrade** command for WebSphere Application Server Version 6.1 retrieves the saved configuration that was created by the **WASPreUpgrade** command from the *backupDirectory* that you specified. The **WASPostUpgrade** script for WebSphere Application Server Version 6.1 reads the configuration from this directory to migrate to WebSphere Application Server Version 6.1 and adds all migrated applications into the *app_server_root*/installedApps directory for the Version 6.1 installation.

Authority

To run this command script, your user profile must have *ALLOBJ authority.

Syntax

The script syntax is as follows:

```
WASPostUpgrade backupDirectory
                    [-profileName profile_name]
                    [-scriptCompatibility true | false]
                    [-portBlock port_starting_number]
                    [-backupConfig true | false]
                    [-replacePorts true | false]
                    [-includeApps true | false | script]
                    [-keepAppDirectory true | false]
                    [-appInstallDirectory user_specified_directory]
                    [-traceString trace_spec [-traceFile file_name]]
```

Parameters

The parameters are as follows:

backupDirectory
This is a required parameter. The value *backupDirectory* specifies the name of the directory in which the **WASPreUpgrade** tool stores the saved configuration and files and from which the **WASPostUpgrade** tool reads the configuration and files.

-profileName
This is an optional parameter for migrating to specific profiles in WebSphere Application Server Version 6.1. The value *profile_name* specifies the name of the Version 6.1 profile to which the script migrates your configuration. You must have already created this profile before calling the **WASPostUpgrade** command.

If the -profileName parameter is not specified, the default profile will be used. If no default profile is found, the system will report an error.

-scriptCompatibility
This is an optional parameter used to specify whether or not migration should create the following Version 5.x or 6.0.x configuration definitions:

- Transport
- ProcessDef
- Version 5.x or 6.0.x SSL

instead of the following Version 6.1 configuration definitions:

- Channels
- ProcessDefs
- Version 6.1 SSL

The default is true.

Specify true for this parameter in order to minimize impacts to existing administration scripts. If you have existing **wsadmin** scripts or programs that use third-party configuration APIs to create or modify the Version 5.x or 6.0.x configuration definitions, for example, you might want to specify true for this option during migration.

Note: This is meant to provide a temporary transition until all of the nodes in the environment are at the Version 6.1 level. When they are all at the Version 6.1 level, you should perform the following actions:

1. Modify your administration scripts to use all of the Version 6.1 settings.
2. Use the **convertScriptCompatability** command to convert your configurations to match all of the Version 6.1 settings.

   See “convertScriptCompatibility command” on page 88.

-portBlock
This is an optional parameter. The *port_starting_number* value specifies the first of a block of consecutive port numbers to assign when the command script runs.

-backupConfig
This is an optional parameter used to specify whether the existing WebSphere Application Server Version 6.1 configuration is saved before any changes are made by the **WASPostUpgrade** tool. The default is true—that is, to use the **backupConfig** command to save a copy of the current configuration into the *profile_name*/temp directory.

Use the **restoreConfig** command to restore that configuration as required.

-replacePorts
This is an optional parameter used to define how to migrate virtual host and server transport ports. By default, migrating adds configuration data from the previous version to the existing data in the WebSphere Application Server Version 6.1 configuration. In some cases, existing port definitions from the earlier release were carefully set to avoid port conflicts with other products. In such cases, it is likely that you would want to migrate the settings into Version 6.1. Use the -replacePorts parameter to totally replace settings in the Version 6.1 environment with the settings from the previous version. The default is false—that is, do not replace default port definitions. Select true to replace all virtual host alias port settings during migration. Transport settings in existing servers are replaced by the settings from the previous version.

-includeApps
This is an optional parameter that can be specified in the following ways:

- True

  Include user enterprise applications as part of the migration.

  This is the default.

- False

  Do nothing with user enterprise applications during **WASPostUpgrade** processing.

- Script

  Prepare user enterprise applications for installation in the WebSphere Application Server Version 6.1 installableApps directory without actually installing them during **WASPostUpgrade** processing.

  JACL scripts that can be used to install these applications are generated and saved in the *backupDirectory* directory. You can then run these files at any point and in any combination after the **WASPostUpgrade** command has completed. You can also reorganize and combine these JACL files for better applications installation efficiency if you want.

WebSphere Application Server system applications will migrate regardless of the value set by this parameter.

-keepAppDirectory
This is an optional parameter used to specify whether to install all applications to the same directories in which they are currently located. The default is false.

If this parameter is specified as true, each individual application retains its location.

Restrictions: If this parameter is specified as true, the location is shared by the existing WebSphere Application Server Version 5.x or 6.0.x installation and the Version 6.1 installation. If you keep the migrated applications in the same locations as those of the previous version, the following restrictions apply:

- The WebSphere Application Server Version 6.1 mixed-node support limitations must be followed. This means that the following support cannot be used when evoking the **wsadmin** command:
  - Precompile JSP
  - Use Binary Configuration

– Deploy EJB

- You risk losing the migrated applications unintentionally if you later delete applications from these locations when administering (uninstalling for example) your Version 5.x or 6.0.x installation.

-appInstallDirectory
This is an optional parameter used to pass the directory name to use when installing all applications during migration. The default of *profile_name*\installedApps is used if this parameter is not specified.

Quotes must be used around the directory name if one or more blanks are in the name.

-traceString
This is an optional parameter. The value *trace_spec* specifies the trace information that you want to collect.

To gather all trace information, specify `"*=all=enabled"` (with quotation marks).

If you do not specify the -traceString or -traceFile parameter, the command creates a trace file by default and places it in the *backupDirectory*/logs directory.

If you specify this parameter, you must also specify the -traceFile parameter.

-traceFile
This is an optional parameter. The value *file_name* specifies the name of the output file for trace information.

If you do not specify the -traceString or -traceFile parameter, the command creates a trace file by default and places it in the *backupDirectory*/logs directory.

If you specify the -traceString parameter but do not specify the -traceFile parameter, the script does not generate a trace file.

Logging

Using the utilities CD-ROM to migrate product configurations

The WebSphere Application Server Version 6.1 utilities CD-ROM can be used to run the **WASPreUpgrade** command against an existing Version 5.x or 6.0.x environment.

See “Overview of migration, coexistence, and interoperability” on page 72 and “Premigration considerations” on page 73.

Use it for migration involving operating systems that are not supported by WebSphere Application Server Version 6.1.

1. Run the **WASPreUpgrade** command on the utilities CD-ROM against the existing WebSphere Application Server version on the existing operating-system level.
   - Specify the migration backup directory.
   - Specify the name of the instance or profile root directory for the current WebSphere Application Server Version 5.x instance or 6.0.x profile.

   See “WASPreUpgrade command” on page 90.
2. Upgrade the operating system.
3. Install WebSphere Application Server Version 6.1.
4. Run the **WASPostUpgrade** command manually.
   See “WASPostUpgrade command” on page 91.

Tip: For help in troubleshooting problems when migrating, see Chapter 11, “Troubleshooting migration,” on page 115.

Migrating Cloudscape databases

After you use the migration tools to migrate to WebSphere Application Server Version 6.1, you should verify the results of the automatic Cloudscape database migration and manually migrate any Cloudscape database instances that are not automatically migrated by the tools.

See “Overview of migration, coexistence, and interoperability” on page 72 and “Premigration considerations” on page 73.

Tip: Before you run the migration tools, ensure that the *debug migration trace* is active. By default, this trace function is enabled. To reactivate the debug migration trace if it is disabled, set one of the following trace options:

- `all traces*=all`
- `com.ibm.ws.migration.WASUpgrade=all`

WebSphere Application Server Version 6.1 requires Cloudscape Version 10.1.

Cloudscape Version 10.1 is a pure Java database server that combines the Apache Derby runtime with the opportunity to use the full services of IBM Software Support. For comprehensive information about Cloudscape Version 10.1, see the Cloudscape section of ibm.com at http://www.ibm.com/software/data/cloudscape/.

For help in troubleshooting problems when migrating, see Chapter 11, “Troubleshooting migration,” on page 115.

1. Verify the automatic migration of Cloudscape database instances.

   When you migrate from WebSphere Application Server Version 5.x or 6.0.x to Version 6.1, the migration tools automatically upgrade the database instances that are accessed through the embedded framework by some internal components such as the UDDI registry. The tools also attempt to upgrade Cloudscape instances that your applications access through the embedded framework. You must verify these migration results after running the migration tools.

   See “Verifying the Cloudscape v10.1.x automatic migration” on page 45.

2. Manually migrate Cloudscape database instances where necessary.

   The Version 6.1 migration tools do not attempt to automatically migrate database instances that transact with applications through the Cloudscape Network Server framework. This exclusion eliminates the risk of corrupting third-party applications that access the same database instances as those accessed by WebSphere Application Server.

   For details on manually migrating database instances that are accessed through the Cloudscape Network Server framework as well as Cloudscape instances that fail the automatic migration, see “Upgrading Cloudscape manually” on page 47.

3. Manually migrate your UDDI registry if it uses a database on the Cloudscape Network Server framework.

   See “Migrating the UDDI registry” on page 38.

Chapter 6. Migrating Web server configurations

You can migrate a Web server from supporting an earlier version of WebSphere Application Server to support the current version.

1. Configure an HTTP server instance.

   There are two options from which to choose:

   - Create a new HTTP server instance to be used by the WebSphere Application Server Version 6.1 profile.

     This method allows WebSphere Application Server Version 5.x or 6.0.x and Version 6.1 profiles to continue operating correctly.
   - Update the HTTP server instance configuration for the WebSphere Application Server Version 5.x or 6.0.x profile that is being migrated.

     This method changes the HTTP instance configuration to work with the WebSphere Application Server Version 6.1 profile and makes the WebSphere Application Server Version 5.x or 6.0.x profile no longer usable.
2. Configure the virtual host for the WebSphere Application Server Version 6.1 profile.

   This step ensures that both the host and HTTP transport port number exist in the virtual host list.

   If you created a new HTTP server in the previous step or if you used the -portBlock parameter when performing the migration, the virtual host will not contain the correct port for communication with your HTTP server. You need to add a host alias for the port used by your HTTP server.
3. Configure communication with Web servers.

   This step regenerates the plug-in configuration file, plugin-cfg.xml. It needs to be done after any configuration changes have been made.

   Additional configuration is required if Secure Sockets Layer (SSL) is enabled on a plug-in transport. In addition to copying the .kdb file to the Version 6.1 profile, you must edit the plug-in to specify the .kdb file required for the plug-in to use the transport.

   For more information on copying the .kbd files to the Version 6.1 profile, see the section on J2EE security in “Configuration mapping during product-configuration migration” on page 78.

Note on the applicability of the information in this section: This information in this section is only applicable if you migrated from WebSphere Application Server Version 5.x; it is not applicable if you migrated from Version 6.0.x.

The plug-in configuration file (plugin-cfg.xml) generated after successful migration from Version 5.x to Version 6.1 is topology centric—that is, it includes all the applications within a cell. You can manage this cell-wide plug-in configuration file from the Version 6.1 administrative console, by using the GenPluginCfg command, or by using the Plug-in Config Generator MBean.

Be aware that regenerating the plug-in configuration can overwrite manual configuration changes that you might want to preserve.

The application-centric generation of the plugin-cfg.xml file is supported using the Version 6.x administrative console. Being application centric means that the plugin-cfg.xml file generated in the administration console has a granularity that allows each application to be mapped to its specific Web or application server.

To set up the administrative console so that you can use it to manage the Web server plug-in configuration, you must first create a default Web server configuration and then use the administrative console to add the plug-in properties from your migrated plugin-cfg.xml file to this Web server configuration.

- To create a default Web server configuration and then add the plug-in properties from your migrated plugin-cfg.xml file in a stand-alone application server configuration, perform the following tasks:
  1. Create a default Web server configuration.
  2. Use the Version 6.x administrative console to edit the configuration and define the plug-in properties.

Chapter 7. Migrating administrative scripts

You can migrate administrative scripts using scripting and the wsadmin tool.

WebSphere Application Server Version 6.1 supports migrating administrative scripts from V5.x.

- If you are migrating administrative scripts from V5.x, see:
  "Migrating administrative scripts from 5.x to 6.x"
- If you are migrating administrative scripts from V6.0.2, see:
  "Migrating administrative scripts from V6.0.2 to V6.1" on page 102

Migrating administrative scripts from 5.x to 6.x

There are some changes you should be aware of when migrating from WebSphere Application Server V5.x to V6.x

There are a few changes to be aware of that are required for your existing scripts when moving to WebSphere Application Server Version 6.x. In general, the administration model has changed very little. However, there are some changes required with Version 6.x. See the following steps for the three categories of changes:

- Prepare for evolutionary changes **without** script compatibility support. These types of changes can be evolved directly without the assistance of script compatibility support. Read "Making changes from 5.x to 6.x to accommodate changes without scripting support" on page 100 for more information.
- Prepare for evolutionary changes **with** script compatibility support. The default is to migrate using compatibility mode, where old object types are migrated into the new configuration and all existing scripts will run unchanged. Read "Making changes with scripting support to administrative scripts from 5.x to 6.x" on page 100 for more information.
- Prepare for required changes. Read "Making required changes to administrative scripts migrated from 5.x to 6.x" for more information.

Making required changes to administrative scripts migrated from 5.x to 6.x

There are some required changes you need to make when migrating from WebSphere Application Server V5.x to V6.x

There are a few required changes that you need to make for your existing scripts when moving to WebSphere Application Server Version 6.x. In general, the administration model has changed very little. However, there are some changes required with Version 6.x. See the following steps for making the necessary changes:

- Be aware of the implications of migrating JMS applications from the embedded messaging in WebSphere Application Server Version 5 to the default messaging provider in WebSphere Application Server Version 6.x.
- A new version of Jacl (1.3.2) is shipped with WebSphere Application Server V6.x. With this Jacl version, regexp command supports only tcl 8.0 regexp command syntax. If your existing V5.x Jacl script uses regexp command syntax that is supported in Jacl 1.2.6 but not anymore in Jacl 1.3.2, you may not get a match anymore, or you may get a compile error for your regexp command similar to the following:

```
com.ibm.bsf.BSFException: error while eval'ing Jacl expression:
couldn't compile regular expression pattern: ?+* follows nothing
     while executing
"regexp {(?x)
     ..."
     ("if" test expression)
     invoked from within
```

```
"if {[regexp {(?x)
     ..."
     (file testregexp.jacl line 2)
     (file line 2)
     invoked from within
"source testregexp.jacl"
```

There is no workaround for this regression problem. Jacl has indicated that this is by design and there is no simple patch to fix this design decision.

- For WSADMIN $AdminConfig: The PME CacheInstanceService type is no longer used in Version 6.x. If your scripts contain code to set the **enable** attribute on the CacheInstanceService type, remove the code. It is not needed in Version 6.x.

Making changes from 5.x to 6.x to accommodate changes without scripting support

There are some changes without scripting support you should be aware of when migrating from WebSphere Application Server V5.x to V6.x, as they can affect script compatibility.

There are a few changes to be aware of that are required for your existing scripts when moving to WebSphere Application Server Version 6.x. These types of changes can be evolved directly without the assistance of script compatibility support. The data can be accessed from multiple locations, including the old and new locations. As long as the new location is not updated, the data is accessed from the old location. Once the new location is updated, it becomes the current data and is used for further accesses and updates. Warning messages are logged when the old location is still being used. See the following steps for changes:

Prepare for evolutionary changes **without** script compatibility support.

The location of the `transaction logs directory` attribute has changed from the `ApplicationServer::TransactionService` to the `ServerEntry::recoveryLogs`. As long as the new location is not used, the value from the old location will continue to be used. Scripts that modify the old location can still be used; that value will take effect until a value in the new location is set. The change to scripts to use the new location is as follows:

Old location:

- Using Jacl:

```
set transService [$AdminConfig list TransactionService $server1]
    $AdminConfig showAttribute $transService transactionLogDirectory
```

New Location:

- Using Jacl:

```
$AdminConfig list ServerEntry $node
set serverEntry <select one of the ServerEntry from output of above command>
set recoveryLog [$AdminConfig showAttribute $serverEntry recoveryLog]
$AdminConfig showAttribute $recoveryLog transactionLogDirectory
```

Making changes with scripting support to administrative scripts from 5.x to 6.x

There are some changes that support scripting that you should be aware of when migrating from WebSphere Application Server V5.x to V6.x

There are a few changes to be aware of that are required for your existing scripts when moving to WebSphere Application Server Version 6.x. These changes are assisted by the compatibility mode provided by “WASPostUpgrade command” on page 91. During migration, the default is to migrate using compatibility mode. If this option is taken, then the old object types are migrated into the new configuration; all existing scripts will run unchanged.

The following changes can be made with **with** script compatibility support.

HTTP transports: the new architecture for V6.x uses the new channel framework. HTTP definitions are mapped on top of this support. When compatibility mode is chosen, the old HTTPTransport objects are migrated and mapped onto the channel architecture. Existing scripts can modify these objects and will run unchanged.

Process definition: The name of this object is changed from processDef to processDefs. You can mitigate this change by using the compatibility mode mapping provided by the migration tools. The change to scripts to use the new location is as follows:

Old example:

- Using Jacl:

```
set processDef [$AdminConfig list JavaProcessDef $server1]
set processDef [$AdminConfig showAttribute $server1 processDefinition]
```

  Using Jython:

```
processDef = AdminConfig.list('JavaProcessDef', server1)
print processDef
```

New example. Identify the process definition belonging to this server and assign it to the processDefs variable:

- Using Jacl:

```
set processDefs [$AdminConfig list JavaProcessDef $server1]
set processDefs [$AdminConfig showAttribute $server1 processDefinitions]
```

  Using Jython:

```
processDefs = AdminConfig.list('JavaProcessDef', server1)
print processDefs
```

Example: Migrating - Modifying Web container port numbers

These examples demonstrate how to modify Web container HTTP transport ports for WebSphere Application Server V5.x and V6.x.

Use the following examples:

- wsadmin V5.x

  Using Jacl:

```
set httpPort  7575
set server [$AdminConfig getid /Cell:myCell/Node:myNode/Server:server1/]
set transports [$AdminConfig list HTTPTransport $server]
set transport [lindex $transports 0]
set endPoint [$AdminConfig showAttribute $transport address]
$AdminConfig modify $endPoint [list [list port $httpPort]]
$AdminConfig save
```

  Using Jython:

```
httpPort = 7575
server = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:server1/")
transports = AdminConfig.list("HTTPTransport", server).split(java.lang.System.getProperty("line.separator"))
transport = transports[0]
endPoint = AdminConfig.showAttribute(transport, "address")
AdminConfig.modify(endPoint, [["port", httpPort]])
AdminConfig.save()
```

- wsadmin V6.x

  Using Jacl:

```
set serverNm server1
set newPort  7575
set node [$AdminConfig getid /Cell:myCell/Node:myNode/]
set TCS [$AdminConfig getid /Cell:myCell/Node:myNode/Server:server1/TransportChannelService:/]
set chains [$AdminTask listChains $TCS {-acceptorFilter WebContainerInboundChannel}]

foreach chain $chains {
   set channels [lindex [$AdminConfig showAttribute $chain transportChannels] 0]
   foreach channel $channels {
```

```
      if {[catch {set channelEndPointName [$AdminConfig showAttribute $channel endPointName]} result]} {
         # ignore the error as not all channel has endPointName attribute
      } else {
         set serverEntries [$AdminConfig list ServerEntry $node]
         foreach serverEntry $serverEntries {
            set sName [$AdminConfig showAttribute $serverEntry serverName]
            if {$sName == $serverNm} {
               set specialEndPoints [lindex [$AdminConfig showAttribute $serverEntry specialEndpoints] 0]
               foreach specialEndPoint $specialEndPoints {
                  set endPointNm [$AdminConfig showAttribute $specialEndPoint endPointName]
                  if {$endPointNm == $channelEndPointName} {
                     set ePoint [$AdminConfig showAttribute $specialEndPoint endPoint]
                     $AdminConfig modify $ePoint [list [list port $newPort]]
                     break
                  }
               }
            }
         }
      }
   }
}

$AdminConfig save
```

Using Jython:

```
serverNm = "server1"
newPort = "7575"
node = AdminConfig.getid("/Cell:myCell/Node:myNode/")
TCS = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:server1/TransportChannelService:/")
chains = AdminTask.listChains(TCS, "[-acceptorFilter WebContainerInboundChannel]").split(java.lang.System.getProperty("line.separator"))

for chain in chains:
   channels = AdminConfig.showAttribute(chain, "transportChannels")[1:len(channels)-1].split(" ")
   for channel in channels:
      try:
         channelEndPointName = AdminConfig.showAttribute(channel, "endPointName")
         serverEntries = AdminConfig.list("ServerEntry", node).split(java.lang.System.getProperty("line.separator"))
         for serverEntry in serverEntries:
            sName = AdminConfig.showAttribute(serverEntry, "serverName")
            if sName == serverNm:
               specialEndPoints = AdminConfig.showAttribute(serverEntry, "specialEndpoints")[1:len(specialEndPoints)-1].split(" ")
               for specialEndPoint in specialEndPoints:
                  endPointNm = AdminConfig.showAttribute(specialEndPoint, "endPointName")
                  if endPointNm == channelEndPointName:
                     ePoint = AdminConfig.showAttribute(specialEndPoint, "endPoint")
                     AdminConfig.modify(ePoint, [["port", newPort]])
                     break
      except:
         # ignore the error as not all channel has endPointName attribute

AdminConfig.save()
```

Migrating administrative scripts from V6.0.2 to V6.1

There are some changes you should be aware of when migrating from WebSphere Application Server V6.0.2 to V6.1

There are a few changes to be aware of that are required for your existing scripts when moving to WebSphere Application Server Version 6.1. In general, the administration model has changed very little. However, there are some changes required with Version 6.1. See the following steps for the two categories of changes:

- Be aware of removed features that may have an impact on administration scripts. For more information, see the “Deprecated and removed features” on page 1 article. These may include the following:
  - Support for the Secure Authentication Service (SAS) IIOP security protocol.
  - Support for the Common Connector Framework (CCF).
  - Support for the IBM Cloudscape Version 5.1.x database.
- Be aware of the change required when creating Service Integration Bus objects. For more information about the **createSIBus** command, see the Creating a service integration bus through the command line article.

Related tasks

Chapter 7, “Migrating administrative scripts,” on page 99
You can migrate administrative scripts using scripting and the wsadmin tool.

Chapter 8. Coexisting

You can create an environment in which multiple versions of WebSphere Application Server can run independently on the same system at the same time. A major consideration in coexistence is the avoidance of port conflicts.

Coexisting, as it applies to WebSphere Application Server products, is running a new release of a WebSphere Application Server product on the same machine at the same time as you run an earlier release or running two installations of the same release of a WebSphere Application Server product on the same machine at the same time.

See “Coexistence support.”

This article discusses which coexistence scenarios are supported.

Coexistence support

Coexistence is a state in which multiple installations and multiple nodes from different versions of WebSphere Application Server run independently in the same environment at the same time.

As it applies to WebSphere Application Server products, coexistence primarily refers to the ability of multiple installations of WebSphere Application Server to run independently on the same machine at the same time. Multiple installations include multiple versions and multiple instances of one version. Coexistence also implies various combinations of Web server interaction.

WebSphere Application Server Version 6.1 products can coexist with the following supported versions:
- IBM WebSphere Application Server Version 5.0.x
- IBM WebSphere Application Server Network Deployment Version 5.0.x
- IBM WebSphere Application Server Version 5.1.x
- IBM WebSphere Application Server Network Deployment Version 5.1.x
- IBM WebSphere Application Server Version 6.0.x
- IBM WebSphere Application Server Network Deployment Version 6.0.x

All combinations of Version 5.x products, Version 6.0.x products, and Version 6.1 products can coexist so long as there are no port conflicts.

WebSphere Application Server Version 5.x and Version 6.0.x clients can coexist with Version 6.1 clients.

Table 3. WebSphere Application Server clients Version 5.x, Version 6.0, and Version 6.1 multiversion coexistence scenarios

Installed product	WebSphere Application Server Version 6.1 clients
WebSphere Application Server Version 5.0	Supported
WebSphere Application Server Network Deployment Version 5.0	Supported
WebSphere Application Server Version 5.0.1	Supported
WebSphere Application Server Network Deployment Version 5.0.1	Supported
WebSphere Application Server Version 5.0.2	Supported
WebSphere Application Server Network Deployment Version 5.0.2	Supported
WebSphere Application Server Version 5.1	Supported

Table 3. WebSphere Application Server clients Version 5.x, Version 6.0, and Version 6.1 multiversion coexistence scenarios (continued)

Installed product	WebSphere Application Server Version 6.1 clients
WebSphere Application Server Network Deployment Version 5.1	Supported
WebSphere Application Server Version 6.0	Supported
WebSphere Application Server Network Deployment Version 6.0	Supported
WebSphere Application Server Version 6.0.1	Supported
WebSphere Application Server Network Deployment Version 6.0.1	Supported
WebSphere Application Server Version 6.0.2	Supported
WebSphere Application Server Network Deployment Version 6.0.2	Supported

WebSphere Application Server Version 5.x and Version 6.0.x products can coexist with Version 6.1 products.

Table 4. WebSphere Application Server Version 5.x, Version 6.0, and Version 6.1 multiversion coexistence scenarios

Installed product	WebSphere Application Server Version 5.x		WebSphere Application Server Version 6.0			WebSphere Application Server Version 6.1		
	Application Server	Network Deployment	Application Server	Network Deployment	Express	Application Server	Network Deployment	Express
WebSphere Application Server Version 5.0	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Network Deployment Version 5.0	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server clients Version 5.0	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Version 5.0.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Network Deployment Version 5.0.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server clients Version 5.0.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Version 5.0.2	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Network Deployment Version 5.0.2	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server clients Version 5.0.2	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported

Table 4. WebSphere Application Server Version 5.x, Version 6.0, and Version 6.1 multiversion coexistence scenarios (continued)

Installed product	WebSphere Application Server Version 5.x		WebSphere Application Server Version 6.0			WebSphere Application Server Version 6.1		
	Application Server	Network Deployment	Application Server	Network Deployment	Express	Application Server	Network Deployment	Express
WebSphere Application Server Version 5.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Network Deployment Version 5.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server clients Version 5.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Version 6.0	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Network Deployment Version 6.0	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server clients Version 6.0	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Version 6.0.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Network Deployment Version 6.0.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server clients Version 6.0.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Version 6.0.2	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Network Deployment Version 6.0.2	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server clients Version 6.0.2	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Version 6.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server Network Deployment Version 6.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
WebSphere Application Server clients Version 6.1	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported

In addition to multiversion coexistence, WebSphere Application Server also lets you install multiple times on one machine (multiple installation instances) or install once and have multiple profiles. Multiple Version 6.1 installation instances on one machine include the following combinations:

- Multiple application server profiles from multiple installations of the WebSphere Application Server product
- Multiple application server profiles from a single installation of the WebSphere Application Server product

Chapter 9. Interoperating

WebSphere Application Server Version 6.1 is interoperable with other WebSphere Application Server versions under certain conditions.

See “Overview of migration, coexistence, and interoperability” on page 72 and “Premigration considerations” on page 73.

WebSphere Application Server Version 6.1 is generally interoperable with WebSphere Application Server Version 5.x and Version 6.0.x. However, there are specific requirements to address for each version. In general, you should be at the most recent group PTF level to support interoperability.

1. Follow the required guidelines for Version 5.x.

   Guideline 1:

   Set the following JVM properties:

   ```
   com.ibm.ejs.jts.jts.ControlSet.nativeOnly=false
   com.ibm.ejs.jts.jts.ControlSet.interoperabilityOnly=true
   ```

   For WebSphere Application Server Version 5.0.2, apply this guideline in addition to moving to Version 5.0.2.8.

   Guideline 2:

   Be aware of the level of WebSphere Application Server in which each function you use is supported. Applications that you intend to be interoperable must only use function that is supported by all levels of WebSphere Application Server in the cluster. For example, applications that use the `commonj.timer.TimerManager` resource, which was new in Version 6.0, should not be deployed to a cluster including both Version 5.1 and Version 6.1 servers.

   Guideline 3:

   If you run related cross-domain interoperating applications (one server is in rtp.raleigh.ibm.com and the other is in cn.ibm.com for example), you need to use fully qualified host names (host9.rtp.raleigh.ibm.com instead of just host9 for example) when installing WebSphere Application Server Version 6.1.

   Guideline 4:

   If you want to interoperate WebSphere Application Server Version 6.1 with Version 5.0, you must be at or above the Version 5.0.2.7 level. If you want to interoperate Version 6.1 with Version 5.1, you must be at or above the Version 5.1.1.1 level. Older levels of Version 5.0 and Version 5.1 do not support interoperability with Version 6.1.

2. Upgrade the Software Development Kit (SDK) used to one supported by Version 6.1.x.

   See Recommended fixes for WebSphere Application Server.

This information is dynamic and might be augmented by information in technical articles that are available on the IBM DeveloperWorks WebSphere site. Check the site for the latest information.

Chapter 10. Configuring ports

When you configure WebSphere Application Server resources or assign port numbers to other applications, you must avoid conflicts with other assigned ports. In addition, you must explicitly enable access to particular port numbers when you configure a firewall.

For more information about port numbers that your iSeries system currently uses, enter the NETSTAT *CNN command on the command line. Press **F14** to view assigned port numbers.

You can also use the port validator tool to find port conflicts between different WebSphere Application Server profiles, products, and servers. See the information center.

1. Review the port number settings, especially when you are planning to coexist.

   You can use the **dspwasinst** command-line tool to display the port information for a profile. See the information center.
2. **Optional:** Change the port number settings.

   You can set port numbers when configuring the product after installation.
   - During profile creation using the **manageprofiles** command, you can accept the default port values or you can specify your port settings. If you want to specify ports, you can do so in any of the following ways:
     - Specify the use of a port file that contains the port values.
     - Specify the use of a starting port value.
     - Specify the use of the default port values.
   - You can use the **chgwassvr** command to change the ports for an application server within a profile.

Port number settings in WebSphere Application Server versions

You should be able to identify the default port numbers used in the various versions of WebSphere Application Server so that you can avoid port conflicts if you plan for an earlier version to coexist or interoperate with Version 6.1.

When you configure WebSphere Application Server resources or assign port numbers to other applications, you must avoid conflicts with other assigned ports. In addition, when you configure a firewall, you must explicitly enable access to particular port numbers.

Notes:

- When you install WebSphere Application Server, the default instance is created with the default port values. When you create a WebSphere Application Server instance with the crtwasinst script, you can specify different port values.
- For more information about port numbers that your iSeries system currently uses, enter the NETSTAT *CNN command on the CL command line. Press F14 to view assigned port numbers.
- You can also use the port validator tool to find port conflicts between different WebSphere Application Server profiles, products, and servers. See the *Using the administrative clients* PDF for more information.

Version 6.1 port numbers

Table 5. Port definitions for WebSphere Application Server Version 6.1

Port Name	Default Value	Files
HTTP_Transport Port (WC_defaulthost)	9080	serverindex.xml and virtualhosts.xml
Administrative Console Port (WC_adminhost)	9060	
HTTPS Transport Port (WC_defaulthost_secure)	9443	
Administrative Console Secure Port (WC_adminhost_secure)	9043	
Bootstrap Port (BOOTSTRAP_ADDRESS)	2809	serverindex.xml
SOAP Connector Port (SOAP_CONNECTOR_ADDRESS)	8880	
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	9401	
CSIV2 Server Authentication Port (CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS)	9403	
CSIV2 Multi-authentication Listener Port (CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS)	9402	
ORB Listener Port (ORB_LISTENER_ADDRESS)	9100	
High Availability Manager Communication Port (DCS_UNICAST_ADDRESS)	9353	
Service Integration Port (SIB_ENDPOINT_ADDRESS)	7276	
Service Integration Port Secure (SIB_ENDPOINT_SECURE_ADDRESS)	7286	
MQ Transport (SIB_MQ_ENDPOINT_ADDRESS)	5558	
MQ Transport secure (SIB_MQ_ENDPOINT_SECURE_ADDRESS)	5578	
SIP Container Port (SIP_DEFAULTHOST)	5060	
SIP Container Secure Port (SIP_DEFAULTHOST_SECURE)	5061	
Internal JMS Server (JMSSERVER_SECURITY_PORT)	5557	
DRS_CLIENT_ADDRESS **Deprecation:** This port is deprecated and is no longer used in the current version of WebSphere Application Server.	7873	
IBM HTTP Server Port	80	virtualhosts.xml, plugin-cfg.xml, and *web_server_root*/conf/ httpd.conf
IBM HTTP Server Administration Port	2001	serverindex.xml

Version 6.0.x port numbers

Table 6. Port definitions for WebSphere Application Server Version 6.0.x

Port Name	Default Value	Files
Web Container Port (WC_defaulthost)	9080	server.xml, plugin-cfg.xml, and virtualhosts.xml
Web Container Secure Port (WC_defaulthost_secure) **Note:** If you change this port number, remember the following information: • To use secure (SSL-enabled) ports you must have the OS/400 or i5/OS Digital Certificate Manager product (5722SS1 option 34) and a Cryptographic Access Provider product (such as 5722AC3) installed. • If you change this port, you must regenerate the Web server plug-in configuration for the application server.	9443	
Administrative Console Port (WC_adminhost)	9060	server.xml and virtualhosts.xml
Administrative Console Secure Port (WC_adminhost_secure)	9043	

Table 6. Port definitions for WebSphere Application Server Version 6.0.x (continued)

Port Name	Default Value	Files
Name Service or RMI Connector Port (BOOTSTRAP_ADDRESS)	2809	serverindex.xml
SOAP Port (SOAP_CONNECTOR_ADDRESS)	8880	
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	9501	
Common Secure Interoperability Version 2 (CSIV2) Server Transport Port (CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS)	9503	
CSIV2 Client Transport Port (CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS)	9502	
Object Request Broker (ORB) Listener Port (ORB_LISTENER_ADDRESS)	Not applicable	
Java Message Service (JMS) Queued Port (JMSSERVER_QUEUED_ADDRESS)	5558	
JMS Direct Port (JMSSERVER_DIRECT_ADDRESS)	5559	
JMS Security Port (JMSSERVER_SECURITY_PORT)	5557	
Data Replication Service Client Port (DRS_CLIENT_ADDRESS) **Deprecation:** This port is deprecated and is no longer used in the current version of WebSphere Application Server.	7873	
IBM HTTP Server Port	80	virtualhosts.xml, plugin-cfg.xml, and *web_server_root*/conf/ httpd.conf
IBM HTTP Server Administration Port	2001	serverindex.xml
Cell Discovery Port (CELL_DISCOVERY_ADDRESS)	Not applicable	
CELL_MULTICAST_DISCOVERY_ADDRESS	Not applicable	
NODE_MULTICAST_IPV6_DISCOVERY_ADDRESS	5001	

Version 5.1 port numbers

Table 7. Port definitions for WebSphere Application Server Version 5.1

Port Name	Default Value	Files
HTTP_TRANSPORT	9080	server.xml and virtualhosts.xml
HTTPS_TRANSPORT	9443	
HTTP_TRANSPORT_ADMIN	9090	
HTTPS_TRANSPORT_ADMIN	9043	
JMSSERVER_SECURITY_PORT	5557	server.xml
JMSSERVER_QUEUED_ADDRESS	5558	serverindex.xml
JMSSERVER_DIRECT_ADDRESS	5559	
BOOTSTRAP_ADDRESS	2809	
SOAP_CONNECTOR_ADDRESS	8880	
DRS_CLIENT_ADDRESS	7873	
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	0	
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	0	
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	0	
ORB_LISTENER_ADDRESS	0	
IBM HTTP Server Port	80	virtualhosts.xml, plugin-cfg.xml, and *web_server_root*/conf/ httpd.conf
IBM HTTPS Server Administration Port	2001	Not applicable

Chapter 11. Troubleshooting migration

You might encounter problems while migrating from an older version of WebSphere Application Server.

- If you encounter a problem when you are migrating from a previous version of WebSphere Application Server to Version 6.1, check your log files and other available information.
  1. Look for the log files, and browse them for clues.
     - *migration_backup_dir*/WASPreUpgrade.*time_stamp*.log
     - *profile_root*/logs/WASPostUpgrade.*time_stamp*.log
     - *app_server_root*/logs/clientupgrade.*time_stamp*.log
  2. Look for `MIGR0259I: The migration has successfully completed.` or `MIGR0271W: The migration completed with warnings.` in the *migration_backup_dir*/WASPreUpgrade.*time_stamp*.log, *profile_root*/logs/WASPostUpgrade.*time_stamp*.log, or *app_server_root*/logs/clientupgrade.*time_stamp*.log.

     If `MIGR0286E: The migration failed to complete.` is displayed, attempt to correct any problems based on the error messages that appear in the log file. After correcting any errors, rerun the command from the bin directory of the product installation root.
  3. Open the Log and Trace Analyzer built into the Application Server Toolkit (AST) on the service log of the server that is hosting the resource that you are trying to access, and use it to browse error and warning messages.
  4. With WebSphere Application Server running, run the **dumpNameSpace** command and pipe, redirect, or ″more″ the output so that it can be easily viewed.

     This command results in a display of all objects in WebSphere Application Server's namespace, including the directory path and object name.
  5. If the object a client needs to access does not appear, use the administrative console to verify the following conditions.
     - The server hosting the target resource is started.
     - The Web module or enterprise Java bean container hosting the target resource is running.
     - The JNDI name of the target resource is properly specified.

  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 IBM Support page. IBM Support also has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.
- During the migration process, problems might occur while you are using the **WASPreUpgrade** tool or the **WASPostUpgrade** tool.
  - Problems can occur when you are using the **WASPreUpgrade** tool.
    - A ″Not found″ or ″No such file or directory″ message is returned from the **WASPreUpgrade** tool.

      This problem can occur if you are trying to run the **WASPreUpgrade** tool from a directory other than the WebSphere Application Server Version 6.1 *app_server_root*\bin. Verify that the **WASPreUpgrade** script resides in the Version 6.1 *app_server_root*\bin directory, and launch the file from that location.
    - The DB2 JDBC driver and DB2 JDBC driver (XA) cannot be found in the drop-down list of supported JDBC providers in the administrative console.

      The administrative console no longer displays deprecated JDBC provider names. The new JDBC provider names used in the administrative console are more descriptive and less confusing. The new providers will differ only by name from the deprecated ones.

      The deprecated names will continue to exist in the jdbc-resource-provider-templates.xml file for migration reasons (for existing JACL scripts for example). However, you are encouraged to use the new JDBC provider names in your JACL scripts.

- You receive the following message when running the **WASPreUpgrade** tool: `MIGR0108E: The specified WebSphere directory does not contain a WebSphere version that can be upgraded.`

  The following possible reasons for this error exist:
  - If WebSphere Application Server Version 5.x or 6.0.x is installed, you might not have run the **WASPreUpgrade** tool from the bin directory of the Version 6.1 installation root.
    1. Look for something like the following message to display when the **WASPreUpgrade** tool runs: `IBM WebSphere Application Server, Release 5.0.`

       This message indicates that you are running the WebSphere Application Server Version 5.0 migration utility, not the Version 6.1 migration utility.
    2. Alter your environment path or change the current directory so that you can launch the WebSphere Application Server Version 6.1 **WASPreUpgrade** tool.
  - An invalid directory might have been specified when launching the **WASPreUpgrade** tool.

  See "WASPreUpgrade command" on page 90.

– Problems can occur when you are using the **WASPostUpgrade** tool.
- A ″Not found″ or ″No such file or directory″ message is returned from the **WASPostUpgrade** tool.

  This problem can occur if you are trying to run the **WASPostUpgrade** tool from a directory other than the WebSphere Application Server Version 6.1 *app_server_root*\bin. Verify that the **WASPostUpgrade** script resides in the Version 6.1 *app_server_root*\bin directory, and launch the file from that location.
- You receive the following message: `MIGR0102E: Invalid Command Line. MIGR0105E: You must specify the primary node name.`

  The most likely cause of this error is that WebSphere Application Server Version 5.x or 6.0.x is installed and the **WASPostUpgrade** tool was not run from the bin directory of the Version 6.1 installation root.

  To correct this problem, run the **WASPostUpgrade** command from the bin directory of the WebSphere Application Server Version 6.1 installation root.
- You receive message: `MIGR0108E: The specified WebSphere directory does not contain WebSphere version that can be upgraded.`

  The following possible reasons for this error exist:
  - If WebSphere Application Server Version 5.x or 6.0.x is installed, you might not have run the **WASPostUpgrade** tool from the bin directory of the Version 6.1 installation root.
    1. Look for something like the following message to display when the **WASPostUpgrade** tool runs: `IBM WebSphere Application Server, Release 5.0.`

       This message indicates that you are running the WebSphere Application Server Release 5.0 migration utility, not the Version 6.1 migration utility.
    2. Alter your environment path or change the current directory so that you can launch the WebSphere Application Server Version 6.1 **WASPostUpgrade** tool.
  - An invalid directory might have been specified when launching the **WASPreUpgrade** tool or the **WASPostUpgrade**.
  - The **WASPreUpgrade** tool was not run.
- You receive the following error: `MIGR0253E: The backup directory migration_backup_directory does not exist.`

  The following possible reasons for this error exist:
  - The **WASPreUpgrade** tool was not run before the **WASPostUpgrade** tool.
    1. Check to see if the backup directory specified in the error message exists.
    2. If not, run the **WASPreUpgrade** tool.

       See "WASPreUpgrade command" on page 90.
    3. Retry the **WASPostUpgrade** tool.
  - An invalid backup directory might be specified.

    For example, the directory might have been a subdirectory of the Version 5.x or 6.0.x tree that was deleted after the **WASPreUpgrade** tool was run and the older version of the product was uninstalled but before the **WASPostUpgrade** tool was run.

1. Determine whether or not the full directory structure specified in the error message exists.
2. If possible, rerun the **WASPreUpgrade** tool, specifying the correct full migration backup directory.
3. If the backup directory does not exist and the older version it came from is gone, rebuild the older version from a backup repository or XML configuration file.
4. Rerun the **WASPreUpgrade** tool.

- You decide that you need to run **WASPreUpgrade** again after you have already run the **WASPostUpgrade** command.

  During the course of a deployment manager or a managed node migration, **WASPostUpgrade** might disable the old environment. If after running **WASPostUpgrade** you want to run **WASPreUpgrade** again against the old installation, you must run the migrationDisablementReversal.jacl script located in the old *app_server_root*/bin directory. After running this JACL script, your Version 5.x or 6.0.2 environment will be in a valid state again, allowing you to run **WASPreUpgrade** to produce valid results.

- Version 6.1 applications fail to install.

  Manually install the applications using the **wsadmin** command after **WASPostUpgrade** has completed.

  To manually install an application that failed to install during migration, use the **wsadmin** command to run the install_*application_name*.jacl script that the migration tools created in the backup directory.

  Use the following parameters:

  ```
  app_server_root/bin/wsadmin -f migration_backup_directory/install_application_name.jacl -conntype NONE
  ```

See "WASPostUpgrade command" on page 91.

- If you select the option for the migration process to install the enterprise applications that exist in the Version 5.x or Version 6.0.x configuration into the new Version 6.1 configuration, you might encounter some error messages during the application-installation phase of migration.

  The applications that exist in the Version 5.x or Version 6.0.x configuration might have incorrect deployment information—usually, invalid XML documents that were not validated sufficiently in previous WebSphere Application Server runtimes. The runtime now has an improved application-installation validation process and will fail to install these malformed EAR files. This results in a failure during the application-installation phase of **WASPostUpgrade** and produces an ″E:″ error message. This is considered a ″fatal″ migration error.

  If migration fails in this way during application installation, you can do one of the following:

  – Fix the problems in the Version 5.x or Version 6.0.x applications, and then remigrate.
  – Proceed with the migration and ignore these errors.

    In this case, the migration process does not install the failing applications but does complete all of the other migration steps.

    Later, you can fix the problems in the applications and then manually install them in the new Version 6.1 configuration using the administrative console or an install script.

If you did not find your problem listed, contact IBM support.

Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories infer specific default directory locations. This topic describes the conventions in use for WebSphere Application Server.

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server products or components, of course, require multiple locations.

app_server_root - the install_root for WebSphere Application Server
The default installation root directory for WebSphere Application Server is the /QIBM/ProdData/WebSphere/AppServer/V61/Base directory.

profile_root
The default directory for a profile named *profile_name* for WebSphere Application Server is the /QIBM/UserData/WebSphere/AppServer/V61/Base/profiles/*profile_name* directory.

app_server_user_data_root - the user_data_root for WebSphere Application Server
The default user data directory for WebSphere Application Server is the /QIBM/UserData/WebSphere/AppServer/V61/Base directory.

plugins_root
The default installation root directory for Web server plug-ins is the /QIBM/ProdData/WebSphere/Plugins/V61/webserver directory.

plugins_user_data_root
The default Web server plug-ins user data root is the /QIBM/UserData/WebSphere/Plugins/V61/webserver directory.

plugins_profile_root
The default Web server plug-ins profile root is the /QIBM/UserData/WebSphere/Plugins/V61/webserver/profiles/*profile_name* directory.

app_client_root
The default installation root directory for the J2EE WebSphere Application Client is the /QIBM/ProdData/WebSphere/AppClient/V61/client directory.

app_client_user_data_root
The default J2EE WebSphere Application Client user data root is the /QIBM/UserData/WebSphere/AppClient/V61/client directory.

app_client_profile_root
The default J2EE WebSphere Application Client profile root is the /QIBM/UserData/WebSphere/AppClient/V61/client/profiles/*profile_name* directory.

web_server_root
The default web server path is /www/*web_server_name*.

shared_product_library
The shared product library, which contains all of the objects shared by all Version 6.1 installations on the system, is QWAS61. This library contains objects such as the product definition, the subsystem description, the job description, and the job queue.

product_library
The product library, which contains program and service program objects (similar to .exe, .dll, .so objects for Windows, Linux, and UNIX operating system platforms), is: QWAS61*x* where *x* is A, B, C, ... For the default installation, the value is QWAS61A.

product_lib
The product library that contains the service program objects for the web server plugins. For the

default Web Server Plugins install, this is QWAS61A. If you install the Web Server Plugins multiple times, the product_lib is QWAS61c, where *c* is B, C, D, ... The plugins_install_root/properties/product.properties contains the value for the product library..

cip_app_server_root
The default installation root directory is the /QIBM/ProdData/WebSphere/AppServer/V61/Base/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

A CIP is a WebSphere Application Server product bundled with optional maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts.

cip_user_data_root
The default user data root directory is the /QIBM/UserData/WebSphere/AppServer/V61/Base/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

cip_profile_root
The default profile root directory is the /QIBM/UserData/WebSphere/AppServer/V61/Base/cip/*cip_uid*/profiles/*profile_name* directory for a customized installation package (CIP) produced by the Installation Factory.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

For trademark attribution, visit the IBM Terms of Use Web site (http://www.ibm.com/legal/us/).