



IBM Software Group

Understanding ClassLoaders WebSphere 5.1, 6.0 and 6.1

Speaker: **Paul Van Norman**



WebSphere® Support Technical Exchange



Agenda

- Classloader overview
- Classloader delegation mode & policies
- Shared Libraries
- Class reloading
- Common Exceptions & Causes

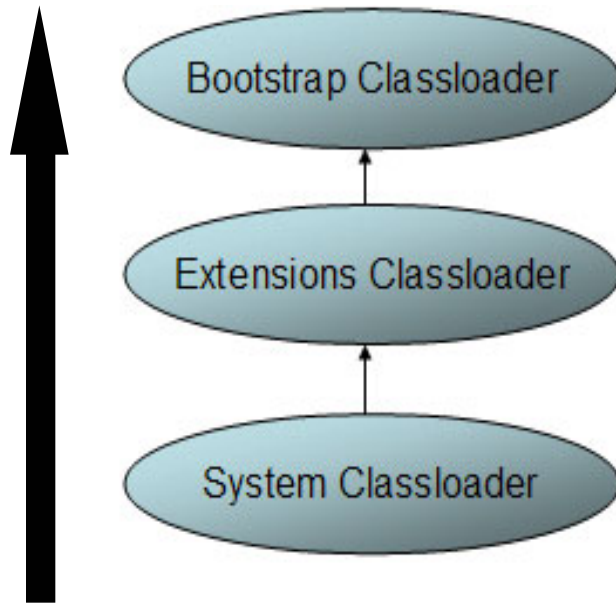


ClassLoaders – Theory of operation

- Java classloader perform three major operations
 - ▶ Load Java class files
 - ▶ Load resource files
 - ▶ Locate native code shared libraries (.dll,.so)
- Java classloaders are an unit of isolation
 - ▶ com.ibm.MyClass loaded by [ClassLoader 1](#) is not equal to com.ibm.MyClass loaded by [ClassLoader 2](#).
 - ▶ A ClassCastException will occur if you try to cast the same type loaded by different classloaders
- Code reloading can occur at a classloader level
 - ▶ To “reload” a class, you must loose all references to any class and objects of a class loaded by a particular classloader, then that classloader can be garbage collected.
 - ▶ Individual classes cannot be reloaded (exception: JDK 1.4 jvms allow this in debug mode).



Java ClassLoader Overview



- **Implemented by NATIVE code**
- Classloader = null
- <JAVA_HOME>/lib
- **sun.misc.Launcher\$ExtClassLoader**
- java.ext.dirs
- <JAVA_HOME>/lib/ext
- **sun.misc.Launcher\$AppClassLoader**
- java.class.path
- CLASSPATH

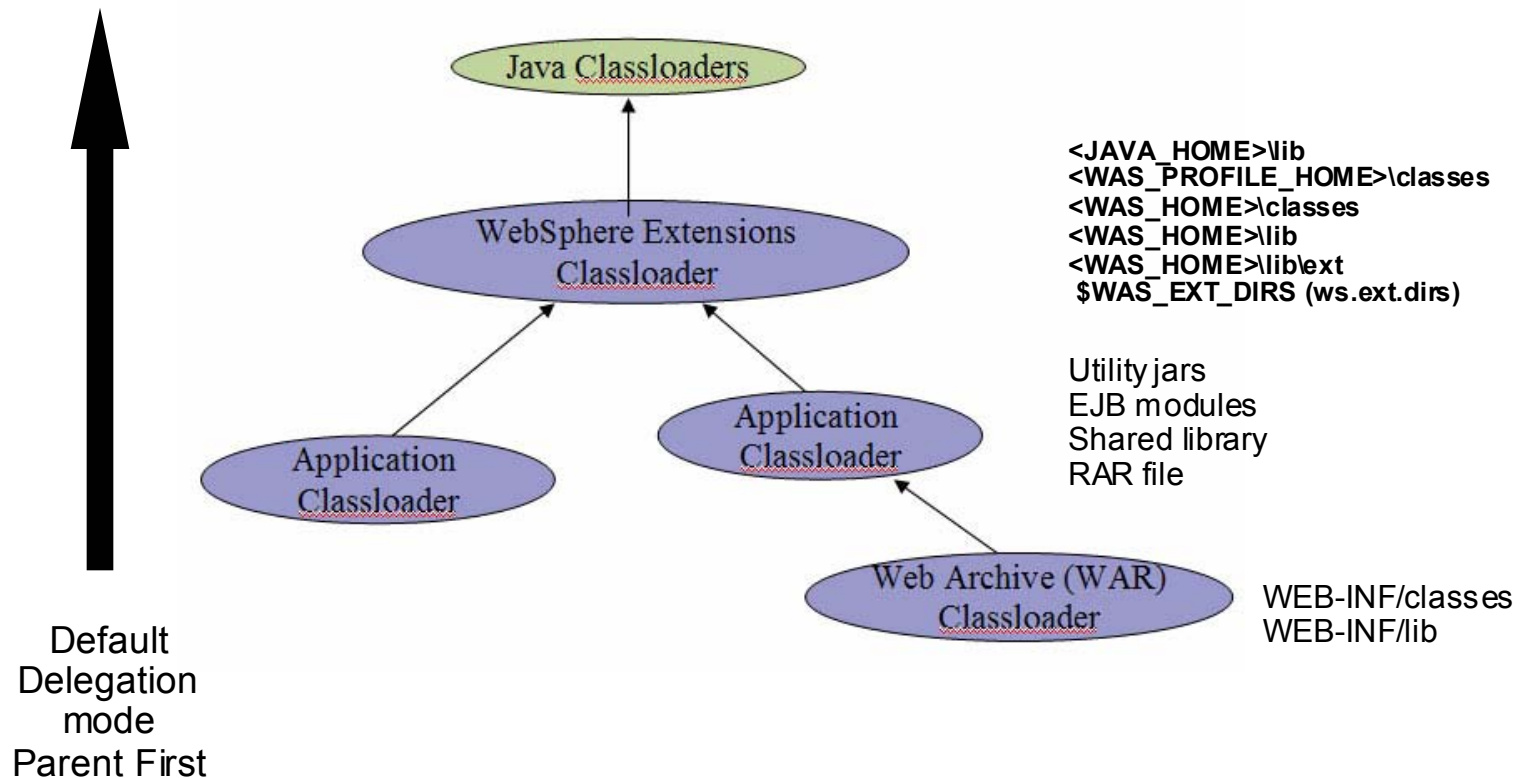
Default Delegation mode:

Parent First

JVM Classloaders

- Bootstrap classloader
 - ▶ Used to load the jar files in <JAVA_HOME>/lib plus extensions specified via **-Xbootclasspath**
 - ▶ getClassLoader() on a class loaded by the bootstrap classloader will return **null** since implemented in **native code**
 - ▶ Typically used to load java.* package classes
- Extensions classloader implemented by **sun.misc.Launcher\$ExtClassLoader**
 - ▶ Used to load extensions to the JVM runtime usually loaded from <JAVA_HOME>/lib/ext but also includes directories specified by the system property **java.ext.dirs**
 - ▶ Used to load security packages, etc.
- CLASSPATH classloader implemented by **sun.misc.Launcher\$AppClassLoader**
 - ▶ Used to load from the directories and jar files specified by the **CLASSPATH** environment variable or **-classpath** command line argument
 - ▶ WebSphere sets following as CLASSPATH
Classpath = d:\Program Files\IBM\WebSphere61\AppServer\profiles\AppSrv01\properties;
d:/Program Files/IBM/WebSphere61/AppServer/properties;
d:/Program Files/IBM/WebSphere61/AppServer/lib/startup.jar;
d:/Program Files/IBM/WebSphere61/AppServer/lib/bootstrap.jar;
d:/Program Files/IBM/WebSphere61/AppServer/lib/j2ee.jar;
d:/Program Files/IBM/WebSphere61/AppServer/lib/improxy.jar;
d:/Program Files/IBM/WebSphere61/AppServer/lib/urlprotocols.jar;
d:/Program Files/IBM/WebSphere61/AppServer/deploytool/itp/batchboot.jar;
d:/Program Files/IBM/WebSphere61/AppServer/deploytool/itp/batch2.jar;
d:/Program Files/IBM/WebSphere61/AppServer/java/lib/tools.jar

WebSphere 5.1 & 6.0 Runtime ClassLoader Overview

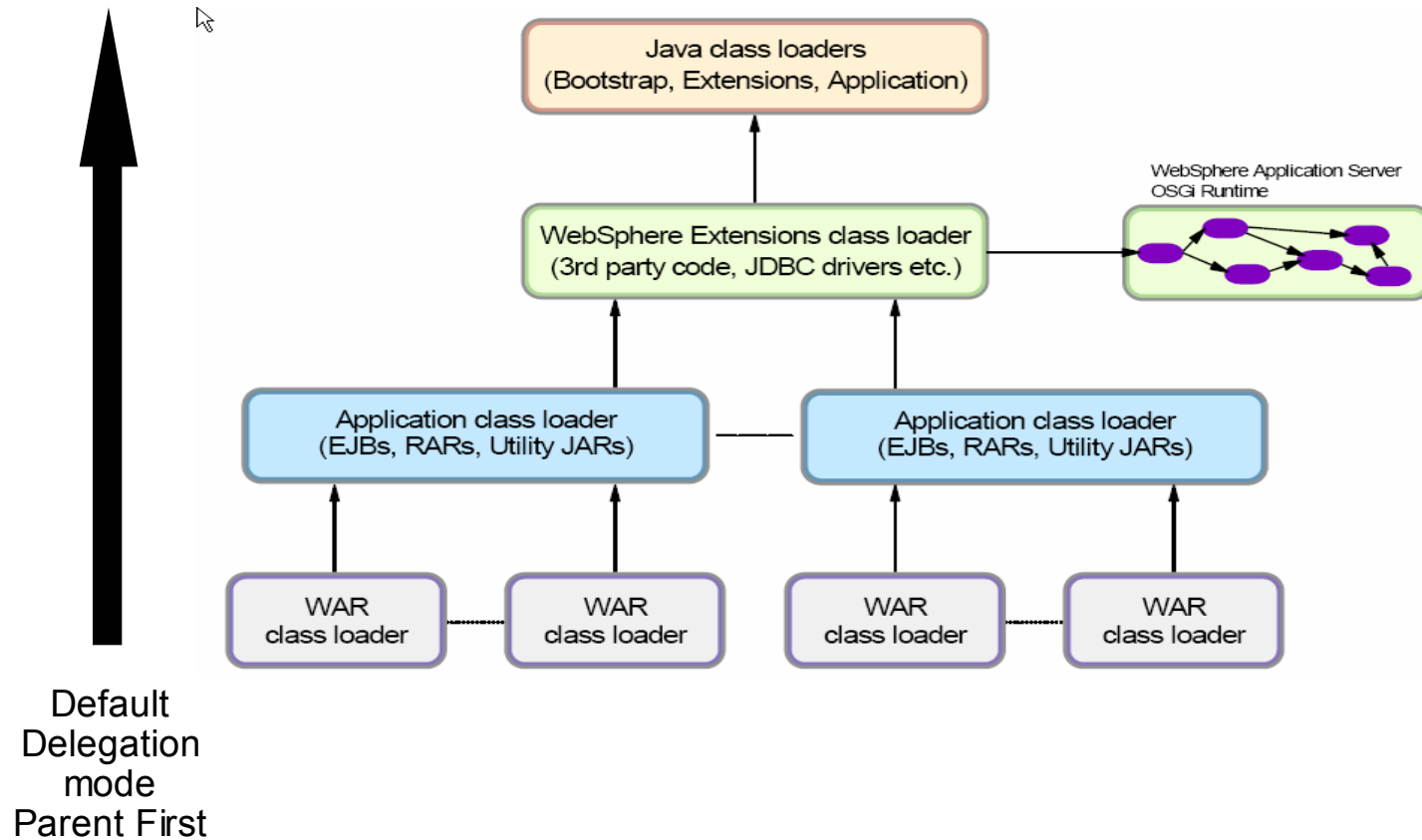


WebSphere ClassLoader Overview

- There are three major “classes” of ClassLoaders in the WebSphere system
 - ▶ System classloader which are provided by the JVM
 - ▶ WebSphere Runtime classloaders which are used to load the WebSphere runtime and some supporting libraries for application use
 - ▶ Application classloaders which are used to load the application artifacts (Web Modules, EJB modules, Utility jars)



WebSphere V6.1 Runtime ClassLoader Overview



WebSphere V6.1 Runtime ClassLoader Overview

- The WebSphere extensions class loader is where WebSphere itself is loaded.
- WebSphere is packaged as a set of OSGi bundles.
 - ▶ <http://www.osgi.org/>
- Each OSGi bundle is loaded separately by its own classloader.
- This network of OSGi class loaders is then connected to the extensions class loader and the rest of the class loader hierarchy through an OSGi gateway classloader.



WebSphere Runtime ClassLoader

- The WebSphere runtime classloader is implemented using the class **com.ibm.ws.bootstrap.ExtClassLoader** (bootstrap.jar)
- The WebSphere runtime is loaded by this classloader by using the directories and jar files present in the directories specified by the system property **ws.ext.dirs** (WAS_EXT_DIRS in setupCmdLine)

```
ws.ext.dirs =  
d:/Program Files/IBM/WebSphere61/AppServer/java/lib;  
d:/Program Files/IBM/WebSphere61/AppServer/profiles/AppSrv01/classes;  
d:/Program Files/IBM/WebSphere61/AppServer/classes;  
d:/Program Files/IBM/WebSphere61/AppServer/lib;  
d:/Program Files/IBM/WebSphere61/AppServer/installedChannels;  
d:/Program Files/IBM/WebSphere61/AppServer/lib/ext;  
d:/Program Files/IBM/WebSphere61/AppServer/web/help;  
d:/Program Files/IBM/WebSphere61/AppServer/deploytool/itp/plugins/com.ibm.etools.ejbdeploy/runtime
```

- Debug information on this classloader can be obtained by using the system property - Dws.ext.debug=true
- Application code should not be loaded using the WebSphere Runtime classloader

Protecting Apps calling 6.1 internal classes

- Ability to restrict access to internal WebSphere classes
- Should not call classes in `com.ibm.ws.*` packages
- Should only call classes in `com.ibm.websphere.*` packages.
- This setting is a per-server (JVM) setting called **“Access to internal server classes”**
- Default value is **“Allow”**

[Application servers](#) > server1

Use this page to configure an application server

Runtime Configuration

General Properties

Name
server1

Node Name
pravindpatel1Node02

☐ Run in development mode

☒ Parallel start

Access to internal server classes
Allow

Server-specific Application Settings

ClassLoader policy
Multiple

Class loading mode
Parent first

Apply OK Reset Cancel

[Application servers](#) > server1

Use this page to configure an application server

Runtime Configuration

General Properties

Name
server1

Node Name
pravindpatel1Node02

☐ Run in development mode

☒ Parallel start

Access to internal server classes
Restrict

Server-specific Application Settings

ClassLoader policy
Multiple

Class loading mode
Parent first

Apply OK Reset Cancel

Application ClassLoader – Delegation Mode

- There are two possible values for a classloader mode:
- **PARENT_FIRST**
 - The PARENT_FIRST classloader mode causes the classloader to first delegate the loading of classes to its parent classloader before attempting to load the class from its local classpath. This is the default for classloader policy and for standard JVM classloaders.
- **PARENT_LAST**
 - The PARENT_LAST classloader mode causes the classloader to first attempt to **load classes from its local classpath before delegating the classloading to its parent**. This policy allows an application classloader to override and provide its own version of a class that exists in the parent classloader.



Application ClassLoader – Web Module Policy

- Another key setting that controls the operation of application classloader is the Web Module Policy
- **Module**
 - ▶ each web module in your application will be loaded by a separate classloader whose parent is the application classloader
- **Application**
 - ▶ web module in your application will be loaded by the single application classloader. Every web module will be able to see every other web module's classes.



Application ClassLoader Policy

- Another key setting that controls the operation of application classloader is the Application Policy
- **Single**
 - ▶ Application classloader **can be shared** by multiple applications
 - ▶ **All Application loaded by one classloader**
- **Multiple**
 - ▶ Application classloader **can not be shared** by multiple applications
 - ▶ Each Application loaded by separate classloader



Application ClassLoader

- Enterprise Application EAR contains
 - ▶ Web modules, EJB modules, application client modules, resource adapters (RAR files), and dependency or utility JARs
- Application classloader loads EJB modules, dependency JAR files, resource adapters, and **shared libraries**.
- Implemented by **com.ibm.ws.classloader.CompoundClassLoader**
 - ▶ <WAS_HOME>/plugins/com.ibm.ws.runtime_6.1.0.jar
 - ▶ <WAS_HOME>/lib/classloader.jar
- Depending on the application classloader policy, an application classloader can be shared by multiple applications (SINGLE) or unique for each application (**MULTIPLE**).
- The application classloader policy controls the isolation of applications running in the system. When set to SINGLE, applications are not isolated. *When set to MULTIPLE, applications are isolated from each other.*
- The application classloader policy is set via the admin console
Servers->Application Servers->Server1

V6.1 Server level Classloading settings

[Application servers](#) > server1

Use this page to configure an application server. An application server

Runtime Configuration

General Properties

Name
server1

Node Name
pravindpatel1Node02

☐ Run in development mode

☒ Parallel start

Access to internal server classes
Allow

Server-specific Application Settings

Classloader policy
Multiple

Class loading mode
Parent first

Apply OK Reset Cancel

[Application servers](#) > server1

Use this page to configure an application server. An applicatio

Runtime Configuration

General Properties

Name
server1

Node Name
pravindpatel1Node02

☐ Run in development mode

☒ Parallel start

Access to internal server classes
Restrict

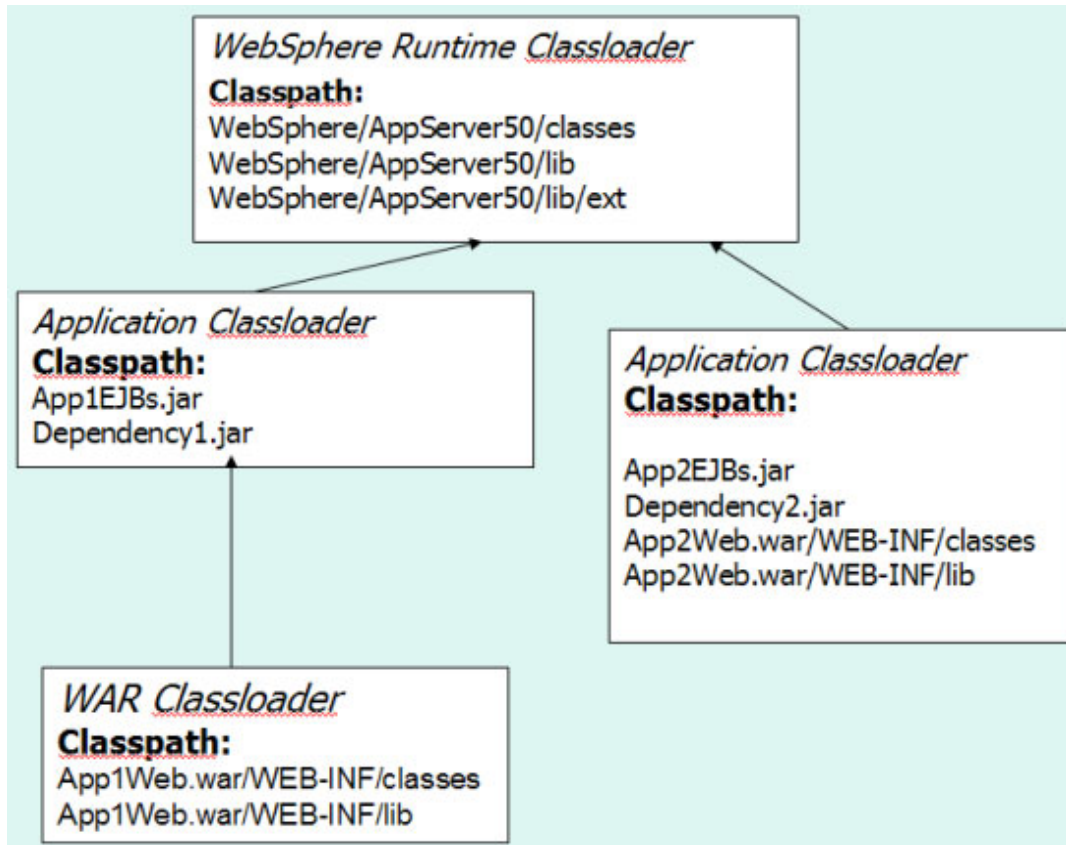
Server-specific Application Settings

Classloader policy
Single

Class loading mode
Parent first

Apply OK Reset Cancel

Application ClassLoader Mode – Multiple (Default)

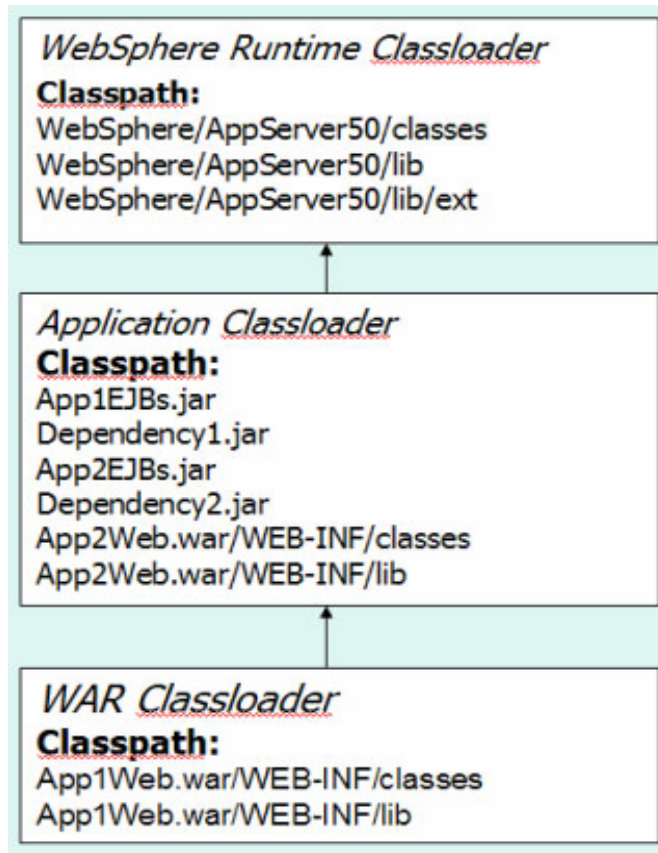


Application
class-loader policy set to
“multiple”

App2Web.war set to
“**Application**” classloader

App1Web.war set to
“**Module**” classloader

Application ClassLoader Mode – Single



Application class-loader policy
set to “**Single**”

All “**Application**” classloader
combined in one due to policy “**Single**”

App2Web.war set to
“**Application**”

App1Web.war set to
“**Module**”

V6.1 Application level Classloading settings

[Enterprise Applications](#) > [DefaultApplication](#) > **Class loader**

Use this page to configure the reloading of classes when application files are updated.

Configuration

General Properties

☐ Reload classes when application files are updated

Polling interval for updated files
 Seconds

Class loader order

☒ Classes loaded with parent class loader first ← PARENT_FIRST

☐ Classes loaded with application class loader first ← PARENT_LAST

WAR class loader policy

☒ Class loader for each WAR file in application ← **Multiple** (separate Application dassloaders)

☐ Single class loader for application ← **Single** (One Application dassloader)

V6.1 WAR Module level Classloading settings

[Enterprise Applications](#) > [DefaultApplication](#) > [Manage Modules](#) > DefaultWebApplication.war

Use this page to configure an instance of a deployed Web module in the application. This page contains deployment and session management settings.

Configuration

General Properties

* URI

DefaultWebApplication.war

Alternate deployment descriptor

* Starting weight

10000

* Class loader order

Classes loaded with parent class loader first

Apply OK Reset Cancel

Addit

[Enterprise Applications](#) > [DefaultApplication](#) > [Manage Modules](#) > DefaultWebApplication.war

Use this page to configure an instance of a deployed Web module in the application. This page contains deployment and session management settings.

Configuration

General Properties

* URI

DefaultWebApplication.war

Alternate deployment descriptor

* Starting weight

10000

* Class loader order

Classes loaded with application class loader first

Apply OK Reset Cancel

Addit

WAR Module Loaded by WAR classloader
“Classes loaded with parent class loader first”


WAR Module Loaded by **Application** classloader
“Classes loaded with application class loader first”

Shared library classloaders – server wide

- Shared libraries enable you to package application code outside the scope of an enterprise application (ear) and have the code visible to either all applications on a server or to specific applications on a server.
- Shared libraries are defined in the admin console using the path Environment->Shared Libraries.
- Like other resources in the console, they can be defined at the Cell, Node, or Server scope and can include variable substitutions.

[Shared Libraries](#) >

New

Specifies a container-wide shared library that can be used by deployed applications. 

Configuration

General Properties	
Name	<input type="text" value="SL1"/>
Description	<input type="text" value="Shared Library one"/>
Classpath	<input type="text" value="\${WAS_HOME}/SL1/sl.jar"/>
Native Library Path	<input type="text"/>

Shared Libraries – server wide - continued

- After defining the shared library, you must associate with a server or application in order to use it.
- To associate it with a server, use the admin console path
Servers->Application Servers->Your Server->ClassLoader to create a new classloader to load the shared library and specify a delegation mode (PARENT_FIRST or PARENT_LAST)
- After creating the classloader, select it and add the library to the classloader

[Application Servers](#) > [server1](#) > [ClassLoader](#) >
New

ClassLoader configuration ⓘ

Configuration	
General Properties	
ClassLoader Mode	PARENT_FIRST ▼
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	

[Application Servers](#) > [server1](#) > [ClassLoader](#) > [ClassLoader_1167935347594](#) > [Library Ref](#) >
New

Library References specify one or more shared libraries used by this application. ⓘ

Configuration	
General Properties	
Library Name	SL1 ▼
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	

ⓘ The name of a shared library has been defined in the shared library configuration documents.

Application ClassLoader - Reloading

- Application classes can be “reloaded” via two methods:
 - Manual** – stop/start an application in the admin console. Do not specify Reload interval.
 - Automatic** – a reload interval can be configured either in the application/web module extensions or at deployment time

Automatic reloading should not be used in production since a considerable overhead is introduced to periodically scan the files that have been used by a classloader to see if any have been modified.

Install New Application

Allows installation of Enterprise Applications and Module

→ Step 1: Provide options to perform the installation

Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	<input type="checkbox"/>
Directory to Install Application	<input type="text"/>
Distribute Application	<input checked="" type="checkbox"/>
Use Binary Configuration	<input type="checkbox"/>
Deploy EJBs	<input type="checkbox"/>
Application Name	<input type="text" value="perfServletApp"/>
Create MBeans for Resources	<input checked="" type="checkbox"/>
Enable Class Reloading	<input checked="" type="checkbox"/>
Reload Interval in Seconds	<input type="text" value="60"/>
Deploy WebServices	<input type="checkbox"/>

Next Cancel

Application packaging suggestions

- Of course, package servlets into their respective WARs and EJBs into their EJB jars. Where to put the rest of the supporting classes?
- For classes only used by one WAR in your application, place them in the WEB-INF/lib directory of your WAR
- For classes shared among multiple modules (EJB or Web), place them in a utility jar at the root of your EAR and reference them using the MANIFEST classpath of each module that will access them
- All EJB modules and Utility jars referenced via the MANIFEST classpath will be loaded using a single Application classloader. This classloader is the parent of the web module classloaders in the application.
- If the Application's ClassLoader policy is Application, then the Web Modules are also loaded by the Application classloader (in addition to EJBs and Utility jars)



ContextClassLoader

- The context classloader is set on a thread by the container (Web or EJB) based on the currently executing application artifact
- The logical classpath is the accumulation of all classpaths searched when a load operation is invoked on a class loader
- The context classloader is set to the top level classloader for the executing application (i.e. the classloader for the web module in the case of a web artifact or the application classloader for an ejb)
- An application can retrieve the context classloader via `Thread.currentThread().getContextClassLoader()`



Common Issues – Native code shared libraries

- Sometimes it is desirable to package code that depends on a native library in an application. JVM will only allow a native library to be loaded **once** into the system since JVM has single address space
- As a result, any class that does a `System.loadLibrary` and has a “native” method cannot be loaded by more than one classloader or reloaded (which would cause it to be loaded by a 2nd classloader)
- The solution is to place such classes in a single instance classloader in the JVM (e.g. `System CLASSPATH` or Server-wide shared library)
- you can break out just the few lines of Java code that load the native code into a class on its own and place this file on WebSphere’s **application class loader** (in a utility JAR)
- If multiple applications (EAR files) deployed to the same application server (JVM), you would have to place the class file on the **WebSphere extensions class** loader instead to ensure the native code is only loaded once per JVM.
- If the native code is placed on a reloadable class loader (such as the application class loader or the WAR class loader), it is important that the native code can properly unload itself should the Java code have to reload.



Common issues – overriding Apache framework

- A common need for applications is to provide a newer (or older) version of the Xerces parser with their application
- If you replace Xerces, then you must also replace Xalan (if in use by your application) or you will get LinkageError: violates loader constraints (see later topic)
- You must use a PARENT_LAST delegation mode to override Xerces so that the application will see the application packaged version instead of the one supplied by WebSphere
- The best way to accomplish this override is to use a server-wide shared library with PARENT_LAST delegation



Common issues – NoClassDefFoundError (NCDFE)

- Thrown if the Java Virtual Machine or a ClassLoader instance tries to load in the definition of a class(as part of a normal method call or as part of creating a new instance using the new expression) i.e. implicit classloading and no definition of the class could be found.
- This exception is thrown when Java code cannot load the specified class.
 - ▶ Invalid or non-existent class
 - ▶ Class path problem
 - ▶ Manifest problem
- Causes and Solutions
 - ▶ The classes have not been placed in the application EAR or elsewhere in the app server classpath. In this case, add the missing classes into the application.
 - ▶ The more interesting scenario is when the classes are present, but still not found. What is happening? As shown earlier, class loading occurs via a parent delegation model, classloading is never delegated to children. So if class X is found higher in the classloader hierarchy, then it cannot find its dependent class Y from a child classloader.
 - ▶ Make sure you package a class and its dependencies at the same level in the classloader hierarchy



Common issues – ClassNotFoundException (CNFE)

- Thrown when an application tries to load in a class through its string name using explicit classloading but no definition for the class with the specified name could be found.
 - The **forName** method in class `Class`.
 - The **findSystemClass** method in class `ClassLoader`.
 - The **loadClass** method in class `ClassLoader`.
- This exception is thrown when Java code cannot load the specified class.
 - ▶ The class is not visible on the logical classpath of the context class loader.
 - The class not found is not in the logical class path of the class loader associated with the **current thread**. The logical classpath is the accumulation of all classpaths searched when a load operation is invoked on a class loader
 - ▶ The application incorrectly uses a class loader API.
 - An application can obtain an instance of a class loader and call either the `loadClass` method on that class loader, or it can call `Class.forName(class_name, initialize, class_loader)` with that class loader. The application may be incorrectly using the class loader application programming interface (API). For example, the class name is incorrect, the class is not visible on the logical classpath of that class loader, or the wrong class loader was engaged..



ClassNotFoundException

► To correct this problem:

- Make the application-specific classes visible to the appropriate application class loader.
- Search for the class not found (Class B).
- If Class B is in the proper location, search for the class that loads the dependent class (Class A) in the Class Load Viewer.
- If the class is loaded and a ClassNotFoundException exception was thrown, then the .jar file or class is not in proper context or the wrong API call in the current context was used. If no class was found, do the following:
 - Search for the class that generated the exception; that is, the class calling `Class.forName`.
 - See which class loader loads the class.
 - Determine whether the class loader has access or can load the class not found by evaluating the class path of the class loader.
- Ensure that the caller class (Class B) is visible to the JVM or WebSphere extensions class loader.



ClassCastException

- Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.
 - ▶ `Object x = new Integer(0);`
 - ▶ `System.out.println((String)x); // x is of superclass type object.`
- The type of the source object is not an instance of the target class (type).
 - ▶ examining the class signature of the source object class, then verifying that it does not contain the target class in its ancestry and the source object class is different than the target class
- The class loader that loaded the source object (class) is different from the class loader that loaded the target class
 - ▶ target class is visible on the classpaths of more than one class loader in the WebSphere Application Server runtime environment.
- The application fails to perform or improperly performs a narrow operation
 - ▶ resolving a remote enterprise bean (EJB) object, the application code does not perform a narrow operation as required.
 - ▶ Do not narrow with super-interface of the EJB instead invoke narrow with the exact EJB interface.

<http://jspwiki.org/wiki/A2AClassCastException>



java.lang.LinkageError – violates loader constraints

- This error occurs when two different views of the same class are defined by the same initiating classloader.
- For example, I have class X that depends on Xerces class A and Xalan class B. Xalan class B also depends on Xerces class A. Suppose we place a version of Xerces class A in our web application with PARENT_LAST delegation and Xerces and Xalan also exist in the WebSphere runtime classloader
- When we load X, then it loads A from the web app and B from the runtime classloader which in turn loads A again from the runtime classloader.
- At this point, the LinkageError is thrown to prevent the potential type spoofing issue.
- Native library problem

<http://www.artima.com/insidejvm/ed2/linkmod20.html>



Class loader viewer

- **Enable the Class loader viewer service:**
 - Used to diagnosing problems with class loaders
 - Enable the class loader viewer service then use the console Class Loader Viewer to examine class loaders and the classes loaded by each class loader.
- **Servers > Application servers > Class Loader Viewer Service.**
- **Troubleshooting > Class Loader Viewer** to access the Class Loader Viewer in the console.

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/utrb_classload_viewer_service.html



Additional WebSphere Product Resources

- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
www.ibm.com/developerworks/websphere/community/
- Learn about other upcoming webcasts, conferences and events:
www.ibm.com/software/websphere/events_1.html
- Join the Global WebSphere User Group Community: www.websphere.org
- Access key product show-me demos and tutorials by visiting IBM Education Assistant:
ibm.com/software/info/education/assistant
- Learn about the Electronic Service Request (ESR) tool for submitting problems electronically:
www.ibm.com/software/support/viewlet/probsub/ESR_Overview_viewlet_swf.html
- Sign up to receive weekly technical My support emails:
www.ibm.com/software/support/einfo.html



Summary

- **ClassLoader overview**
 - ▶ JVM & Websphere classloaders
- **ClassLoader delegation mode & policies**
 - ▶ Parent first/last & application/single
- **Shared Libraries**
 - ▶ Utility jar and native libraries
- **Class reloading**
- **Common Exceptions & Causes**
 - ▶ `ClassNotFoundException`/`ClassCastException`



Questions and Answers

WebSphere Application Server Infocenter:
<http://www-306.ibm.com/software/webservers/appserv/was/library/>

