Session-specific Classpaths – Use Jars Dynamically

BBj® 9.0 introduced session-specific classpath (SSCP) to allow developers to set the Java classpath for BBjSessions on a per-session basis. This capability is especially useful during the development of custom Java code or when the developer wishes to test a third party Java library without restarting BBj Services. This article explores how SSCP positively impacts the development cycle, and more!

Background

Historically, the classpath for BBj Services was configured using the Enterprise Manager. That classpath has acted as the classpath for every BBjProcess within the BBServices process and could not be changed without restarting BBj Services. While BBj Services was running, the classpath could be modified through the Enterprise Manager but the changes would not take effect until BBj Services was restarted. This has meant that when users wrote a BBj program that used embedded Java that was not found in the classpath of the running BBj Services then they would need to add a jar file to the BBj Services classpath and restart BBj Services in order to test their program. Similarly, if custom Java code was modified and the jar file containing that code was updated, the user's program would not see those changes until BBj Services was restarted.

Overview

An SSCP is a classpath that can be used by a BBjSession as an alternative to using the BBj Services classpath. More precisely, an SSCP is a classpath that will be prepended to the BBjSessions classpath to define the classpath for a particular BBjSession. An SSCP is identified using a developer-defined name and can be defined using the Enterprise Manager or can be defined using the BBjAPI. Once an SSCP is defined, a BBjProcess can be started with that SSCP by using command line parameters or by using the BBjAPI method BBjCommandLineObject.setClasspathName(). All of this can be done without restarting BBj Services.

By David Wallwork
Senior Software Architect

Creating an SSCP Using Enterprise Manager

To create a new SSCP, enter the information on the Classpath tab of the Server Information section within Enterprise Manager as shown in **Figure 1**.

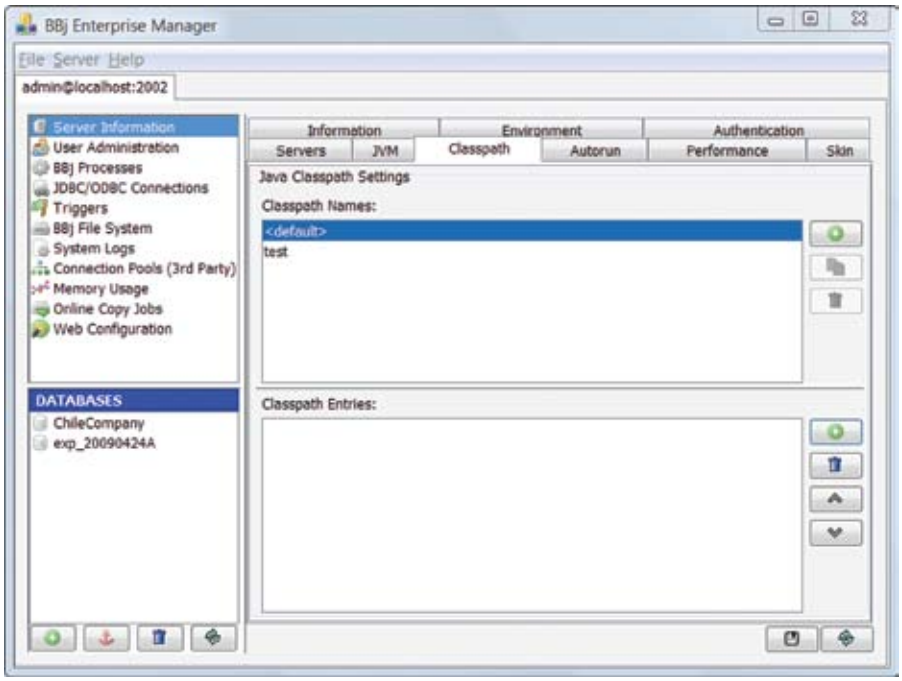

Figure 1. Create a new SSCP

Click the 'add' icon in the 'Classpath Names' pane and then add the desired jar files in the 'Classpath Entries' pane.

Creating an SSCP Using BBjAPI

Alternatively, a developer can create an SSCP programmatically using code similar to **Figure 2**.

```
admin! = BBjAPI().getAdmin(adminName$, adminPassword$)
env! = admin!.getEnvironment()
vec! = BBjAPI().makeVector()
vec!.add("/dev/testSSCP/AFoo.jar")
vec!.add("/dev/testSSCP/BFoo.jar")
env!.setClasspath("test", vec!)
print env!.getClasspath("test")
```

Figure 2. Create an SSCP programmatically

Using an SSCP at the Command Line

Beginning in BBj 9.0, the command line option **-CPclasspathName** specifies which SSCP to use when starting a BBj session. For example, to run the program **ATest.bbj** using the SSCP whose name is **test**, use the command line:

```
bbj -CPtest Atest.bbj
```

Using an SSCP with BBjAPI::newBBjSession()

Another option to using an SSCP within the BBjAPI is with code similar to the example in **Figure 3**. >>

```
config! = BBjAPI().getConfig()
cmd! = config!.getCommandLineObject()
cmd!.setClasspathName("t")
xx = BBjAPI().newBBjSession(cmd!)
```

Figure 3. Using an SSCP within BBjAPI

Modifying an Existing SSCP

Once a BBj session starts with a particular SSCP, changes made to that SSCP will not affect the running BBj session. The code sample in **Figure 4** illustrates this by defining an SSCP named **test** and then starting a BBj session. After starting the BBj session, the SSCP changes, but the BBj session that has already started will not recognize the changes.

```
admin! = BBjAPI().getAdmin("admin","admin123")
config! = BBjAPI().getConfig()
env! = admin!.getEnvironment()
rem set the test SSCP
vec! = BBjAPI().makeVector()
vec!.add("/dev/testSSCP/AFoo.jar")
vec!.add("/dev/testSSCP/BFoo.jar")
env!.setClasspath("test", vec!)
env!.reloadClasspath("test")
rem retrieve a CommandLineObject and set its SSCP
cmd! = config!.getCommandLineObject()
cmd!.setClasspathName("TEST")
rem start a new BBjSession with the TEST SSCP
x = BBjAPI().newBBjSession(cmd!)
rem ================================
rem modify the tests SSCP
rem the changes in test do NOT affect the program
rem that has already been started
rem ================================
vec!.add("/dev/testSSCP/CFoo.jar")
env!.setClasspath("test", vec!)
```

Figure 4. Code sample demonstrating how a running BBj session will not see the changes to its SSCP

Modifying Jar Files and Reloading an SSCP

If an SSCP named **test** contains a jar file named **foo.jar**, and a developer then modifies **foo.jar** on disk, several factors will determine whether a BBjSession using the **test** SSCP will recognize the changes in **foo.jar** (whether files are loaded from the old foo.jar or the newly modified **foo.jar**).

These factors include:

- Whether a program calls `BBjEnvironment::reloadClasspath("test")`
- How the 'Recheck Session Classpath' performance setting in Enterprise Manager was set
- If the BBj session was started before or after the change was made

If any program invokes the method **BBjEnvironment::reloadClasspath("test")**, then all subsequent BBj processes that use the **"test"** SSCP will load classes from the modified jar. It is best to call this method after modifying any jar file that is on any SSCP.

If developers set the Recheck Session Classpath to "development," then BBj checks the date of each file on the SSCP every time they start a new BBj process with that SSCP. This means that any new BBj process that use the **"test"** SSCP will load classes from the modified jar.

It is worth noting that some operating systems, such as Microsoft Windows, do not allow users to modify a file that is in-use. Therefore, when making modifications to custom classes, developers may want to add versioning information to the resultant JAR file name, allowing new BBj sessions to utilize the new JAR immediately.

SSCP and Static Variables

Java shares a static variable across all instances of a given class. In most cases there is only one class for any given class name within the entire JVM so Java shares a static variable across the entire JVM. When using SSCP, this is not always the case. If Foobar.class is found in **foo.jar**, and **foo.jar** appears in two different SSCPs named **testA** and **testB**, then each SSCP will load a different class object for Foobar. This means that the code running within a BBj session with testA SSCP will have a different class object for Foobar than the code running in a BBj session with testB SSCP. Those BBj sessions will see a different value for any static variable that is in Foobar.

Using an environment to reload the testA classpath can cause a similar situation. A BBj session that was started prior to the call to reloadClasspath("testA") and has the command line parameter -CPtestA will have a different SSCP than a BBj session that is started after the call to reloadClasspath("testA") with the command line parameter -CPtestA. Those two sessions will see different values for any static variable that is in Foobar.

SSCP and BASIS Jars

If a class found in a jar on an SSCP has the same fully qualified classname as a class that is found in one of the BASIS jars, then the jar that is on the SSCP is considered to conflict with the BASIS jar. This conflict can cause unpredictable behavior and may result in failure in the user's code. When BBj discovers such a conflict, it generates a nag message alerting the user of the problem (unless the user is running with a DVK license). While the developer can choose to handle this in any of the following ways, the preferred solution is the first:

1. Rename the classes in the user's jar (the one on SSCP) using a utility such as http://code.google.com/p/jarjar/.
2. Ignore both the conflict and the nag message.
3. Ignore the conflict and suppress the nag message by setting the system property **com.basis.bbj.suppressJarConflictNag=true**.

Summary

As the use of embedded Java becomes more pervasive, there is an increased need to allow separate BBj sessions to have individual classpaths so they can access libraries different from those found in the classpath of BBj Services. SSCP provides this capability and can be configured using the BBjAPI or by using Enterprise Manager. Finally, SSCP gives users the flexibility they need during both development and deployment. ■