

Automatic deployment toolkit for an SOA project environment, Part 2: Jython script development for IBM WebSphere Application Server administration

Skill Level: Intermediate

[YuLin Chen \(chenyul@cn.ibm.com\)](mailto:chenyul@cn.ibm.com)

Software Engineer

IBM

[HeQing \(Hawking\) Guan \(guanheq@cn.ibm.com\)](mailto:guanheq@cn.ibm.com)

Staff Software Engineer

IBM

[Qiang Bai \(baiqiang@cn.ibm.com\)](mailto:baiqiang@cn.ibm.com)

Software Engineer

IBM

[YaoFei \(Richard\) Zhu \(zhuyaof@cn.ibm.com\)](mailto:zhuyaof@cn.ibm.com)

Staff Software Engineer

IBM

11 Sep 2008

The [first article](#) in this [series](#) introduced an automatic deployment toolkit (Automatic-DT). This installment, Part 2 of the series, describes Jython, how to develop a Jython script, and how to develop a Jython script on IBM® WebSphere® Application Server. You'll also take a look at a Jython script programming model on WebSphere Application Server V6. Because Jython scripts are essential to the automatic deployment on WebSphere Application Server, this article shows you a general process to develop Jython scripts to manage WebSphere Application Server resources and provides a sample to illustrate this.

What's Jython?

Jython is an implementation of the high-level, dynamic, object-oriented (OO)

language, *Python*. Jython is seamlessly integrated with the Java™ platform. The predecessor to Jython, JPython, is certified as 100% Pure Java. Jython is freely available for both commercial and non-commercial use and is distributed with source code. Jython is fit for embedded scripting, interactive experimentation, and rapid application development. You can write a short but powerful Jython script to replace repetitive tasks. The Jython interactive interpreter makes it convenient to debug applications or experience the functions of the Jython module. Also, it supports a full OO programming model, which makes it a natural fit for Java's OO design.

The latest Jython version is Jython 2.2 RC3. Jython's version numbers represent the version of CPython from which Jython pulls most of its libraries, which are written in pure Python. Compared with Python, the Jython project develops at a slower pace.

The wsadmin tool of WebSphere Application Server V6.0 uses Jython 2.1, which was released in 2001. This version doesn't provide as many modules as in the popular Python 2.5 and higher versions. If you're a Python programmer, you might initially find it awkward to use the wsadmin tool in Jython. But with the help of wsadmin scripting objects and the powerful Java library of WebSphere Application Server, you can easily overcome this.

The version 6.1 release of WebSphere Application Server represents the start of the deprecation process for the Java Tcl (Jacl) syntax that's associated with the wsadmin tool. The Jython syntax for the wsadmin tool is the strategic direction for WebSphere Application Server administrative automation. In the 6.1 release, WebSphere Application Server contains significantly enhanced administrative functions and tooling that support product automation and the use of the Jython syntax.

Note: The remaining issues described in this article are based on WebSphere Application Server V6.0.

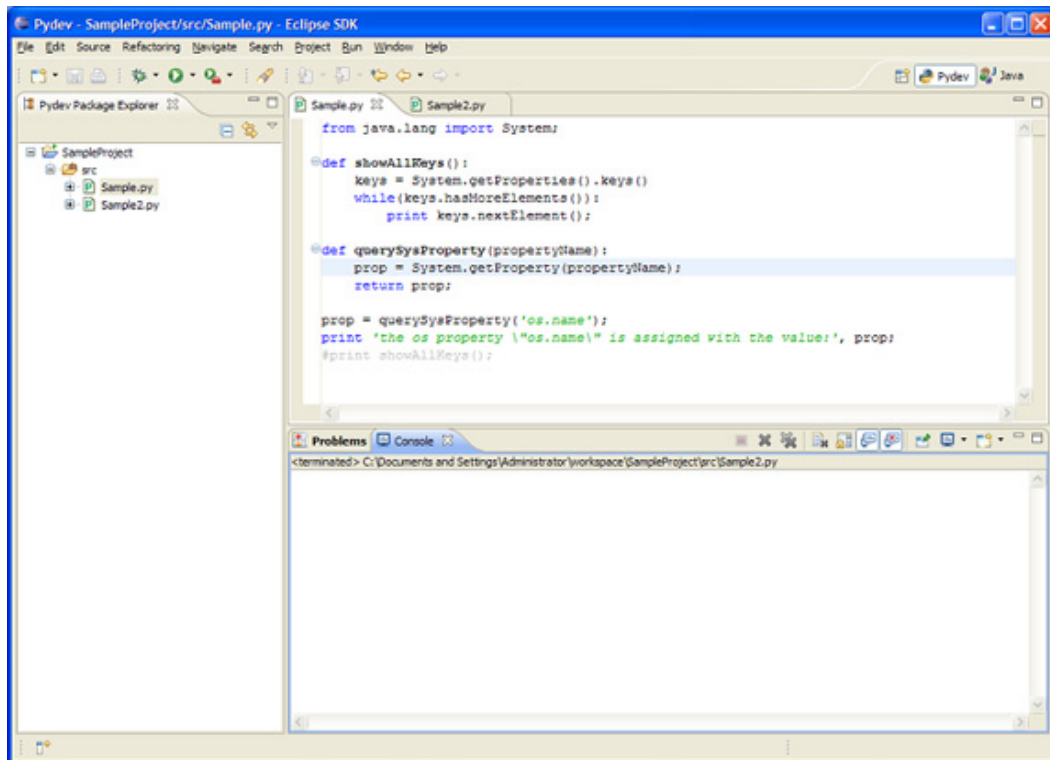
Develop and debug a Jython script

Before you write the first Jython script, you need to prepare the Jython interpreter and editor. A Jython development environment consists of the following components:

- **Jython interpreter:** You can download the Jython interpreter from the [Jython Project home page](#). To run the sample in this article, you need have Jython 2.1 or higher installed on your development system. Considering the compliance with WebSphere Application Server, using the embedded Jython interpreter of wsadmin tool is a good idea.
- **Development system:** This may be any ASCII text editor (such as Microsoft® Windows® Notepad) combined with the command prompt.

The integrated development environment (IDE) tool can provide many helpful features, such as code completion, syntax highlighting, syntax analysis, refactoring, and debugging. PyDev is a plug-in that lets you use Eclipse for Python and Jython development. You can refer to the PyDev Developers Guide to install and configure this plug-in in Eclipse (see [Resources](#) for the link). Figure 1 shows the perspective of PyDev in Eclipse.

Figure 1. PyDev plug-in in Eclipse



- **Java Runtime Environment (JRE):** Jython is a Java implementation of Python, so the JRE is essential for running a Jython script. You need to install the JRE that matches your Jython version. If you select the wsadmin tool as the Jython interpreter, you don't have to install JRE, because the wsadmin tool uses the Java Virtual Machine (JVM) of the local version of WebSphere Application Server.

The first Jython script

After installing the Jython development environment, you can try to write the first Jython script. Suppose that you need to query the system properties, such as the OS platform. The implemented script of such an action is shown in Listing 1.

Listing 1. Example of Jython script

```
#import java.lang.System.class
from java.lang import System

def querySysProperty(propertyName):
    prop = System.getProperty(propertyName)
    return prop

prop = querySysProperty('os.name')
print 'the os property \'os.name\' is assigned with
the value: ', prop
```

Note: In Python, the # symbol is the line comment identifier.

When you input the code from Listing 1 into the Jython interpreter interaction console, the result is shown on the console. You can also save this code fragment as a Jython script file with the extension .py. If the file path is /root/sample.py, use wsadmin to run this script with the command `wsadmin -lang jython -f /root/sample.py`. The result is shown in Listing 2.

Listing 2. Show OS name

```
The os property "os.name" is assigned with the value:
"Linux";
```

If a script invokes the functions or class methods of other Jython modules, use the `execfile` function to load a Jython script from another Jython script file. Listing 3 shows an example:

Listing 3. Invoke an external function

```
execfile('/root/Sample.py')

prop = querySysProperty('os.name')
print 'the os property \'os.name\' is assigned with
the value: ', prop
```

Jython script programming model on WebSphere Application Server

Jython 2.1 has implemented most of the Python 2.1 modules. So the majority of the standard Python documentation applies to Jython. Because the Jython interpreter is implemented by Java, there are many differences, from the trivial to the dramatic, between two Python implementations: Jython and CPython (the common Python implementation). Some built-in extension modules don't exist in Jython, such as `win32com`, `Tkinter`, `socket`, and `logging`. These differences determine that Jython scripts can't run as CPython scripts run, as expected. Because Jython 2.1 was released in early 2001, it doesn't support many popular technologies and industry standards. Despite these shortcomings, you can still develop Jython scripts to complete common computing by Jython modules.

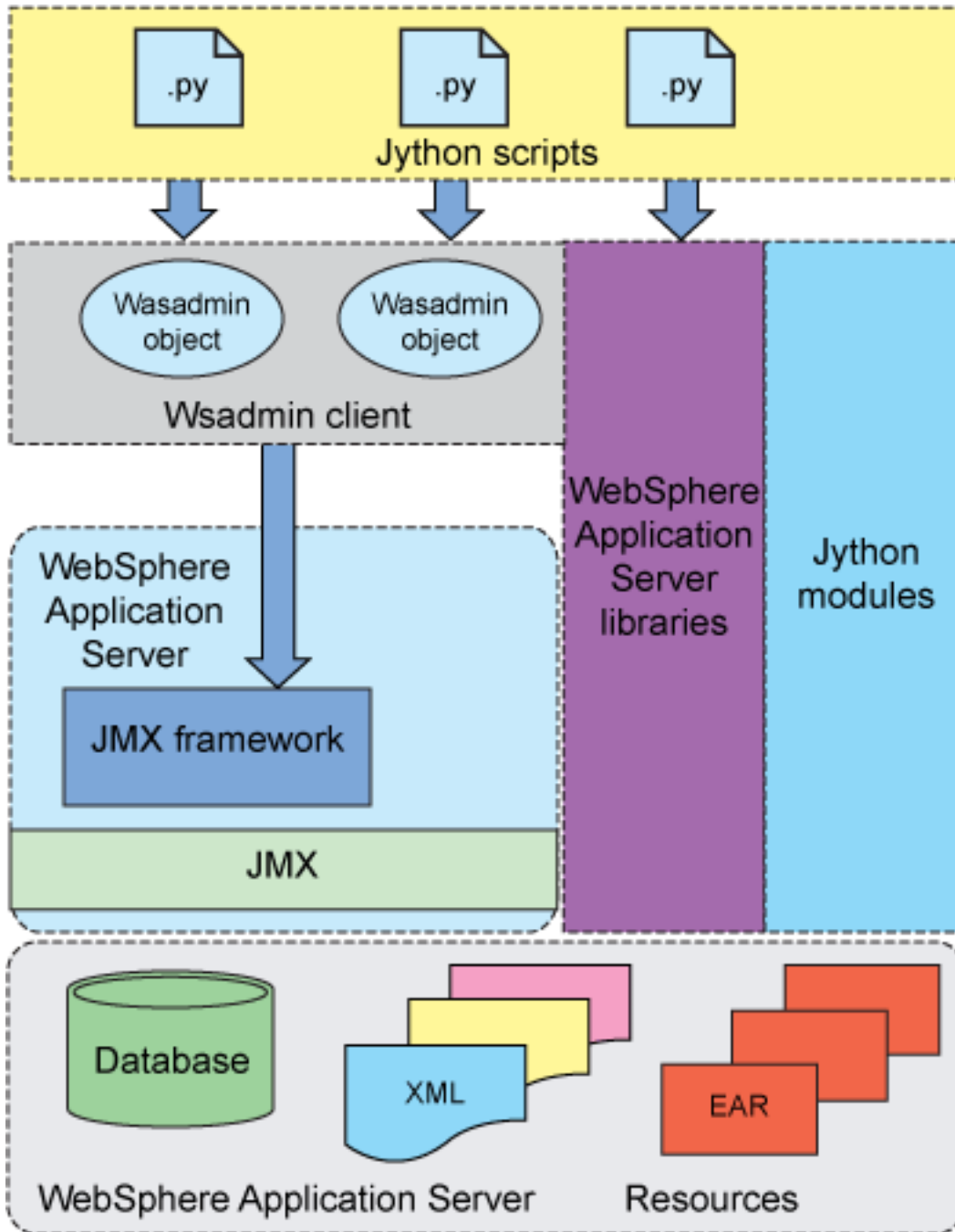
In general, you use Jython scripts to manage the WebSphere Application Server resources to replace the manual work on the WebSphere Application Server Admin Console. You create or delete a resource, modify the configuration of a resource, and read the properties of the resource of interest. For example, you can add or delete a data source, or install or remove an application. WebSphere Application Server V6 has implemented the Java Management Extensions (JMX) framework to manage the resources. However, Jython doesn't provide the functionality to use the JMX framework in its current format. Furthermore, the management of WebSphere Application Server resources is complex. It takes quite a long time to master it and develop a script based only on the Jython modules. Fortunately, you can get enormous help from wsadmin scripting objects and WebSphere Application Server libraries.

WebSphere Application Server provides useful wsadmin scripting objects to enhance the support for Jython. With the help of these scripting objects, a Jython script can take advantage of JMX; create or delete a resource; and query and modify the configurations of WebSphere Application Server resources. Because wsadmin is integrated with the Jython interpreter, wsadmin scripting objects are processed as local objects in Jython scripts, which simplifies the programming greatly.

Another remarkable feature is WebSphere Application Server libraries. As the development of the Jython Project lags behind Python, many emerging technologies—including XML parsers, the use of logging, and Web services—aren't implemented in the current version of Jython. However, many WebSphere Application Server configuration files are properties documents as well as XML documents. In these cases you can leverage the WebSphere Application Server libraries, which are the built-in Java libraries of the WebSphere Application Server. WebSphere Application Server libraries cover all the libraries of Java 2 Platform, Standard Edition (J2SE), and WebSphere Application Server libraries support most of the current industry standards, such as Document Object Model (DOM) and Simple API for XML (SAX). With the help of WebSphere Application Server libraries, you can effectively increase your productivity.

The Jython script programming model highlights the scripting objects and WebSphere Application Server library; it's shown in Figure 2.

Figure 2. Jython script programming model



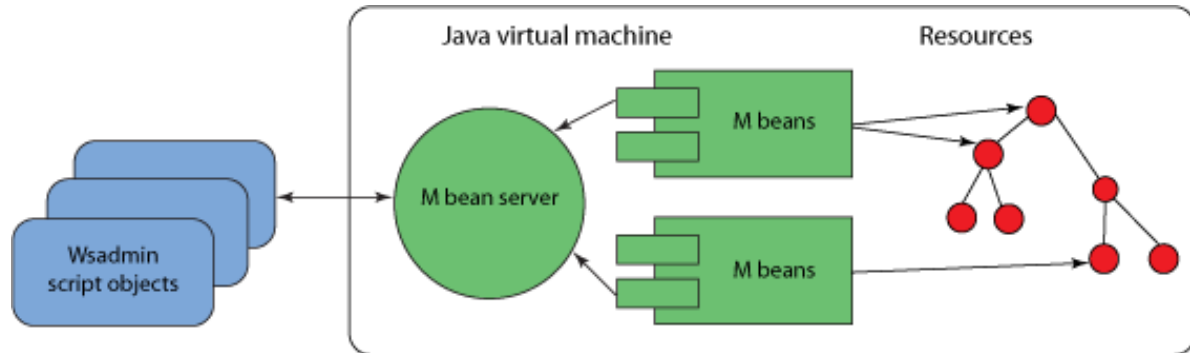
The next section goes into detail about wsadmin scripting objects.

Use wsadmin scripting objects

The wsadmin tool uses a set of management objects that lets you execute commands and command parameters to configure your environments. The scripts use these objects to communicate with MBeans that run in WebSphere Application

Server processes. MBeans are Java objects that represent JMX resources, which is an optional package in addition to J2SE. JMX provides a simple and standard way to manage Java objects, as shown in Figure 3.

Figure 3. wsadmin objects and JMX



The wsadmin tool supports two scripting languages: Jacl and Jython. Five objects are available when you use scripts:

- AdminControl
- AdminConfig
- AdminApp
- AdminTask
- Help

The next two sections introduce you to `AdminControl` and `AdminConfig`, two objects frequently used in Jython scripts for WebSphere Application Server administration.

AdminControl

The `AdminControl` object is used to manage the running instance of an application server by a JMX bean. `AdminControl` works only in the connected mode of the WebSphere Application Server and provides the following functions:

- `AdminControl` retrieves context information, such as Cell and Node.
- Given a name fragment or name template, `AdminControl` can query from the WebSphere Application Server and list all complete names that satisfy the query conditions. The code fragment in Listing 4 gets the first server name in the current context.

Listing 4. Get WebSphere Application Server server name

```
#get the current WAS Cell and Node name.
```

```
cellName = AdminControl.getCell()
nodeName = AdminControl.getNode()

#execute query to get server list in the current
scope.
serverList =
AdminControl.completeObjectName('type=Server,
node=nodeName, cell="' + cellName, '*')

#get the first server name
server = serverList.split(lineSeparator)[0]
```

- `AdminControl` can start or stop the server, get the server status, test the database connection, and so on.

AdminConfig

The `AdminConfig` object is used to run configuration commands to create or modify WebSphere Application Server configuration elements. This scripting object is available in both connected mode and local mode in which the scripting object is not connected to a running server. However, if a server is currently running, you shouldn't run `AdminConfig` in local mode because the configuration changes can't be synchronized with the running server. `AdminConfig` provides lots of methods to query attributes and create, remove, and modify configurations.

`AdminConfig` interacts with the configuration object to change the configurations of WebSphere Application Server. A configuration object is the memory image of the configuration element and is the entry to access and change a resource. Before you create a configuration object, you should know its type. There are many types of configuration objects in WebSphere Application Server V6.0. The command in Listing 5 lists all the types.

Listing 5. All WebSphere Application Server Configuration types

```
print 'The types in this system are listed below:'
print AdminConfig.types()
```

Each type has a number of attributes, which are name-value pairs. Data types of attributes can be:

- String
- Boolean
- int
- ENUM
- types of configuration objects

You can get the attributes information for a type by invoking an attribute method of

AdminConfig. The code fragment in Listing 6 shows attributes of DataSource.

Listing 6. Get the attributes of DataSource

```
print 'The DataSource Type has these attributes:'
print AdminConfig.attributes('DataSource')
```

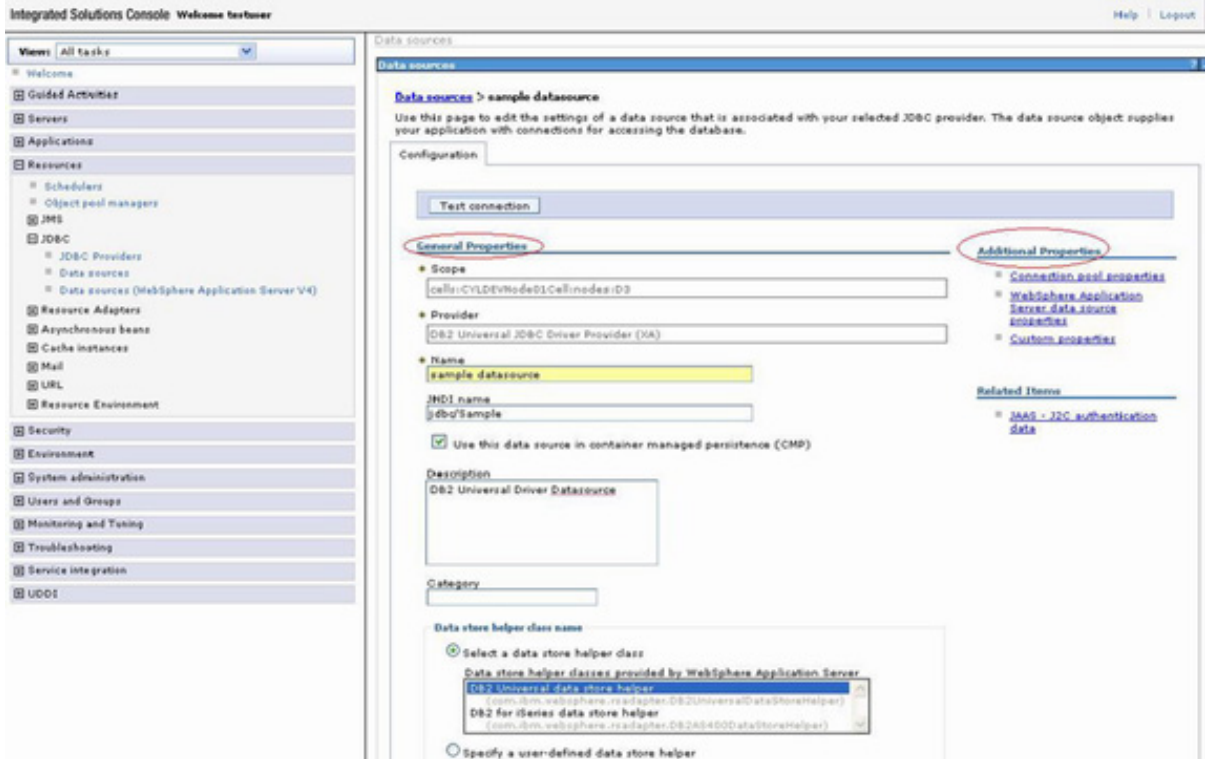
Run this script in the console, and you get the output shown in Listing 7.

Listing 7. Attributes of DataSource

```
DataSource Type has these attributes:
authDataAlias String
authMechanismPreference ENUM(BASIC_PASSWORD, KERBEROS)
category String
connectionPool ConnectionPool
datasourceHelperClassname String
description String
jndiName String
logMissingTransactionContext boolean
manageCachedHandles boolean
mapping MappingModule
name String
preTestConfig ConnectionTest
propertySet J2EEResourcePropertySet
provider J2EEResourceProvider@
providerType String
relationalResourceAdapter J2CResourceAdapter@
statementCacheSize int
xaRecoveryAuthAlias String
```

Many attributes have default values, but only the required attribute must be assigned a value. You can get the required attributes by invoking the required method. However, even if an attribute is not required, its value might affect the behavior of the relational resource. For example, if the mapping attribute of the DataSource object is not set, the data source might not act as the data source in the container-managed persistence (CMP). If you don't know the proper value of an attribute, you can refer to the identical page of the admin console (see Figure 4).

Figure 4. Data source settings in the WebSphere Application Server Admin Console

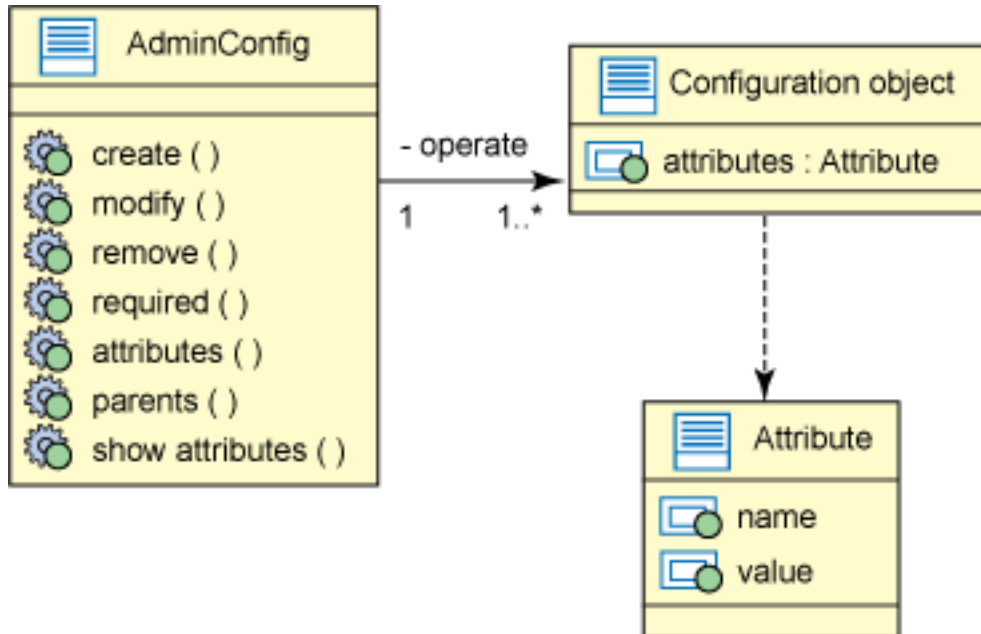


Some attributes, including Scope and JNDI name, appear in the center of the page while some attributes are categorized under Additional Properties on the right. The name of the attribute might not be the same as the setting name, but they should be identical.

To create a new configuration object, you need to provide the parent ID of this object as well as its attributes. You can get the parent type from the resource hierarchy in the Admin Console. Data sources are the children node of a JDBC provider, so its parent is a JDBC provider. Furthermore, you can invoke the parents' method to get the parent hierarchy of the specified type if the resources are based on the XML configuration files. A successful creation call returns the ID of a new configuration object. You can then query attributes of the new configuration object or modify it.

The AdminControl, configuration object, and attribute are shown in Figure 5.

Figure 5. AdminControl, configuration object, and attribute



In general, you can use the following steps to create or modify a configuration object:

1. Identify the configuration type and the corresponding attributes.
2. Create a new configuration, and query the parent ID. Modify a configuration, and query an existing configuration object to get a configuration ID to use.
3. Set the attributes, then create a new configuration object or modify the existing one.
4. Save the configuration.

Listing 8 illustrates how to create and modify a data source.

Listing 8. Create a data source

```

#set the required attributes
parentId = AdminConfig.getid('/Cell:'+ cellName
+'Node:')
nodeName +'/JDBCProvider:DB2XA/');
name = ['name', 'DB2DS']
dsAttrs = [name]
newds = AdminConfig.create('DataSource', parentId,
dsAttrs)
print 'The new DataSource Id: ',newds

#Save the new Object into the configuration
repository.
AdminConfig.save();
  
```

The output is the resource ID, which looks like Listing 9.

Listing 9. the created DataSource ID

```
The new DataSource ID is:
DB2DS1(cells/CYLNode01Cell/nodes/SSD
|resources.xml#DataSource_1194502886875)
```

The string in the parentheses is the XML file path of the resource and the ID of the XML element. When you navigate to the XML document, you'll find this resource in the substring with the # character (as Listing 10 shows).

Listing 10. Data source XML file

```
<resources.jdbc:JDBCProvider
xmi:id="JDBCProvider_1189665535312" name="DB2XA"
providerType="DB2 Universal JDBC Driver Provider (XA)"
implementationClassName="com.ibm.db2.jcc.DB2XADataSource"
xa="true">
<classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar
</classpath>
<classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar
</classpath>
<classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar
</classpath>
<factories xmi:type="resources.jdbc:DataSource"
xmi:id="DataSource_1194502886875" name="DB2DS"
description="Data source template"
authMechanismPreference="BASIC_PASSWORD"
relationalResourceAdapter="builtin_rra">
<connectionPool xmi:id="ConnectionPool_1194502886891"
connectionTimeout="180"
maxConnections="10" minConnections="1" reapTime="180"
unusedTimeout="1800"
agedTimeout="0" purgePolicy="EntirePool"
numberOfSharedPoolPartitions="0"
numberOfUnsharedPoolPartitions="0"
numberOfFreePoolPartitions="0"
freePoolDistributionTableSize="0" surgeThreshold="-1"
surgeCreationInterval="0"
testConnection="false" testConnectionInterval="0"
stuckTimerTime="0"
stuckTime="0" stuckThreshold="0"/>
</factories>
</resources.jdbc:JDBCProvider>
```

Sample: Create a shared library for an application

Let's create a container-wide shared library that's used by deployed applications. When a shared library is referred by an application, WebSphere Application Server creates a configuration object of type `LibraryRef`. You develop a Jython script according to these steps:

1. Set the scope of the shared library; it's one of a Cell, a Node, and a Server.

2. Provide the class path and the library name, and create the shared library.
3. The parent of `LibraryRef` is type `ClassLoader`. So you need to get an object of type `ClassLoader` from the application.
4. Provide the application name and the name of the library, and create a new object of type `LibraryRef`.
5. Save all configurations to complete this task.

Based on these steps, the Jython script is developed as Listing 11 shows.

Listing 11. Jython script to create a shared library

```
import sys

#the class path of shared library, which contains all
necessary jar packages
clsPath = '/pf/p2/lib/MySharedLib'
#the shared library name
libName = 'MySharedLib'
#the name of the application, which references this
shared library
appName = 'query'

nodeName = AdminControl.getNode()
cellName = AdminControl.getCell()
serverList =
AdminControl.completeObjectName('type=Server,node='+\
    nodeName+',cell=' + cellName + ',*')

server = serverList.split(lineSeparator)[0]
serverName = AdminControl.getAttribute(server,'name')
nodeId = AdminConfig.getid('/Cell:'+ cellName
+ '/Node:'+ nodeName+'/')

try:
#create the shared library configuration object
    libId = AdminConfig.create('Library', nodeId,
[['name',libName],\
    ['classPath', clsPath]])
    print "Created shared library:" + libId
except:
    print sys.exc_type,sys.exc_value
    sys.exit()

#get the deployment object of the application
deployId = AdminConfig.getid('/Deployment:' + appName
+ '/')
if(deployId==''):
    print 'This application does not exist:', appName
    sys.exit()

#get the class loader to load the shared library
appDeploy = AdminConfig.showAttribute(deployId,
'deployedObject')
classLoader = AdminConfig.showAttribute(appDeploy,
'classloader')
try:
```

```
#create the library reference object for the
application
refId = AdminConfig.create('LibraryRef',
classLoader,\
    [['libraryName', libName],
['sharedClassLoader', 'true']])
print 'Application '+appName+' has been attached
with this shared library '+libName
except:
    print sys.exc_type,sys.exc_value
    sys.exit()

#save all changes and make them persistent
AdminConfig.save();
```

Run this script and get this result shown in Listing 12.

Listing 12. The created SharedLib ID

```
Created shared
library:MySharedLib(cells/ASRDEVNode02Cell/nodes/p2Node|
libraries.xml#Library_1210649435766)
Application query has been attached shared library
MySharedLib
```

Conclusion

This article introduced Jython, how to develop a Jython script, and the PyDev plug-in for Jython development, all highlighted by developing a Jython script programming model on WebSphere Application Server V6. Using this model, you can take advantage of wsadmin scripting objects and WebSphere Application Server libraries. Finally, you created a sample shared library for an application.

Resources

Learn

- Get complete product information on [WebSphere Application Server](#).
- Get the latest [Jython technical resources and downloads](#).
- Get the latest [Python technical resources and downloads](#).
- Visit the [PyDev Project Web site](#).
- The [SOA and Web services zone](#) on IBM developerWorks hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop Web services applications.
- Play in the [IBM SOA Sandbox!](#) Increase your SOA skills through practical, hands-on experience with the IBM SOA entry points.
- The [IBM SOA Web site](#) offers an overview of SOA and how IBM can help you get there.
- Stay current with [developerWorks technical events and webcasts](#).
- Browse for books on these and other technical topics at the [Safari bookstore](#).
- Check out a quick [Web services on demand demo](#).
- Get an [RSS feed for this series](#). (Find out more about [RSS](#).)

Get products and technologies

- Innovate your next development project with [IBM trial software](#), available for download or on DVD.

Discuss

- [Participate in the discussion forum for this content](#).
- Get involved in the developerWorks community by participating in [developerWorks blogs](#).

About the authors

YuLin Chen

YuLin Chen is a software engineer in the IBM Global Business Solution Center. He is responsible for developing the solution assets built on IBM's Service-Oriented Architecture (SOA) methodology.

HeQing (Hawking) Guan

HeQing Guan (Hawking) is a senior software engineer on the IBM Global Business Solution Center team. He has a doctorate from the Chinese Academy of Sciences and has worked in the SOA field for almost 5 years.

Qiang Bai

Qiang Bai has worked as a software engineer with the Global Business Solution Center in CSDL. In addition to focusing on software configuration management (SCM) products, he is responsible for developing the solution assets built on IBM's Service-Oriented Architecture (SOA) methodology.

YaoFei (Richard) Zhu

YaoFei Zhu (Richard) joined the IBM China Development Lab in 2005 and has worked in Linux on POWER and on Global Business Solution Center (GBSC) teams as a system developer, application developer, and infrastructure architect. He has more than 6 years of experience in AIX, Linux, System p, System x, storage, and SOA.

Trademarks

IBM, the IBM logo, ibm.com, developerWorks, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.