Chapter 8 of *WebSphere Application Server V7 Administration and Configuration Guide,* SG24-7615

WebSphere Application Server V7: Administration with Scripting

The administrative console is sufficient for tasks that are non-repetitive, have a minimal number of administrative steps, and are relatively simple. For administration that requires many steps, which can be repetitive, and time consuming to configure, wsadmin combined with scripts is an ideal tool.

In this chapter we introduce the `wsadmin` scripting solution and describe how you can use it to perform basic tasks.

This chapter contains the following topics:

- "Overview of WebSphere scripting" on page 2
- "Launching wsadmin" on page 3
- "Command and script invocation" on page 7
- "wsadmin management objects" on page 9
- "Managing WebSphere using script libraries" on page 14
- "Assistance with scripting" on page 33
- "Example: Using scripts with the job manager" on page 38
- "Online resources" on page 47

Overview of WebSphere scripting

WebSphere Application Server provides a scripting interface based on the *Bean Scripting Framework (BSF)* called *wsadmin*. BSF is an open source project to implement an architecture for incorporating scripting into Java™ applications and applets. The BSF architecture works as an interface between Java applications and scripting languages. Using BSF allows scripting languages to do the following tasks:

- Look up a pre-registered bean and access a pre-declared bean
- Register a newly created bean
- Perform all bean operations
- Bind events to scripts in the scripting language

Because **`wsadmin`** uses BSF, it can make various Java objects available through language-specific interfaces to scripts. Figure 1 shows the major components involved in the **`wsadmin`** scripting solution.

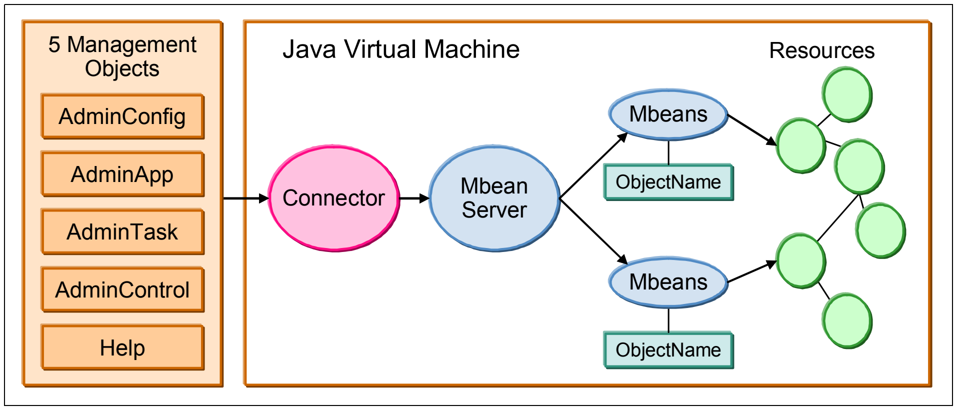


Figure 1 wsadmin scripting

Script programming languages

Two programming language are used to write **`wsadmin`** scripts -- Jython and Jacl. WebSphere® Application Server V7.0 represents the start of the deprecation process for the Jacl syntax. The script library and the command assistance on the administrative console only support Jython.

If you have existing Jacl scripts and want to start migrating to Jython, the Jacl-to-Jython conversion utility can be used to convert the scripts. This conversion assistant typically does 95-98% of a preliminary conversion. In most cases, the resulting conversion is syntactically and runtime equivalent.

However, we strongly recommend that you verify each line to ensure that the code functions as you originally intended. When Jacl and Jython language differences can result in lines of code that are difficult to convert automatically, the converted lines are flagged `#?PROBLEM?`. This provides assistance in finding areas that are most likely in need of manual conversion.

Launching wsadmin

The **`wsadmin`** command file resides in the bin directory of every profile. Start **`wsadmin`** from a command prompt with the command:

- (UNIX®) *`profile_root`*`/bin/wsadmin.sh`
- (Windows®) *`profile_root`*`\bin\wsadmin`

Note that the **`wsadmin`** command also exists in the bin directory of the *install_root* directory. If you start **`wsadmin`** from this location, you must be careful to specify the profile to work with in the command. If you do not specify the profile (or forget to specify it), the default profile will be chosen.

Example 1 illustrates how to start wsadmin. In this example, the wsadmin command is used to connect to the job manager. It is issued from the bin directory of the job manager profile, so the profile does not need to be specified. The -lang argument indicates Jython will be used (Jacl is the default).

Example 1   flexible management: wsadmin command-line

```
C:\WebSphereV7\AppServer\profiles\jmgr40\bin>wsadmin -lang jython
WASX7209I: Connected to process "jobmgr" on node jmgr40node using SOAP
connector
;  The type of process is: JobManager
WASX7031I: For help, enter: "print Help.help()"
wsadmin>
```

To get syntax-related help, use **`wsadmin -?`** or **-help** (see Example 2).

Example 2   wsadmin syntax

```
wsadmin
        [ -h(elp)  ]
        [ -?  ]
        [ -c <command> ]
        [ -p <properties_file_name>]
        [ -profile <profile_script_name>]
        [ -f <script_file_name>]
        [ -javaoption java_option]
        [ -lang  language]
        [ -wsadmin_classpath  class path]
        [ -profileName profile]
        [ -conntype
                SOAP
                        [-host host_name]
                        [-port port_number]
                        [-user userid]
                        [-password password] |
                RMI
                        [-host host_name]
                        [-port port_number]
                        [-user userid]
                        [-password password] |
                JSR160RMI
                        [-host host_name]
                        [-port port_number]
                        [-user userid]
                        [-password password] |
                IPC
                        [-ipchost host_name]
                        [-port port_number]
                        [-user userid]
                        [-password password] |
                NONE
        ]
        [ -jobid <jobid_string>]
        [ -tracefile <trace_file>]
        [ -appendtrace <true/false>]
        [ script parameters ]
```

Scripting environment properties file

The properties that determines the scripting environment for **`wsadmin`** can be set using either the command line or a properties file. Modifying the properties file can be useful when you want to change a default setting, for example, changing the language from Jacl to Jython.

Properties can be set in the following locations:

- The installation default properties file for the profile located at:

  profile_root/properties/wsadmin.properties

- A user default properties file located in the Java user.home property.

- A customized properties file placed in the location pointed to by the WSADMIN_PROPERTIES environment variable.
- A customized properties file pointed to by using the -p argument to the **`wsadmin`** command.

When wsadmin is started, properties are loaded from these files in the order listed above. The properties file that is loaded last overrides the ones loaded earlier.

The properties are listed in Table 1.

Table 1 wsadmin properties

Property	Value
`com.ibm.ws.scripting.connectionType`	SOAP, RMI or NONE
`com.ibm.scripting.port`	TCP port of target system
`com.ibm.scripting.host`	Host name of target system
`com.ibm.ws.scripting.defaultLang`	Jython or Jacl
`com.ibm.ws.scripting.echoparams`	Determines whether parameters or arguments are output to STDOUT or to the wsadmin trace file
`com.ibm.ws.scripting.traceFile`	File for trace information
`com.ibm.ws.scripting.validationOutput`	Location of validation reports
`com.ibm.ws.scripting.traceString`	=com.ibm.*=all=enabled
`com.ibm.ws.scripting.appendTrace`	Appends to the end of the existing log file
`com.ibm.ws.scripting.profiles`	List of profiles to be run before running user commands, scripts, or an interactive shell
`com.ibm.ws.scripting.emitWarningForCustomSecurityPolicy`	Controls whether message WASX7207W is emitted when custom permissions are found
`com.ibm.ws.scripting.tempdir`	Stores temporary files when installing applications
`com.ibm.ws.scripting.validationLevel`	Level of validation to use when configuration changes are made from the scripting interface

Property	Value
com.ibm.ws.scripting.crossDocumentValidationEnabled	Determines whether the validation mechanism examines other documents when changes are made to one document
com.ibm.ws.scripting.classpath	List of paths to search for classes and resources

Some of the listed properties in the **wsadmin**.properties file are commented out by default. An example is `com.ibm.ws.scripting.traceString`. If you want to trace **wsadmin** execution, remove the comment sign # from the properties file.

Some of the properties contain default values. For example, com.ibm.ws.scripting.connectionType has a default value of SOAP. This means when a scripting process is invoked, a SOAP connector is used to communicate with the server. The com.ibm.ws.scripting.defaultLang property is set to Jacl.

Example: specifying a properties file (-p)

Use the -p option to specify a customized properties file. Example 3 shows sample coding for invoking **wsadmin** to execute a script file using a specific properties file.

Example 3 specifying properties file on the command line

```
C:\WebSphereV7\AppServer\profiles\dmgr40\bin>wsadmin -p
c:\webspherev7\appserver
\profiles\dmgr40\properties\wsadmin_custom.properties
WASX7209I: Connected to process "dmgr" on node dmgr40node using SOAP
connector;
 The type of process is: DeploymentManager
WASX7031I: For help, enter: "print Help.help()"
```

Script profile file

A script profile is a script that is invoked before the main script or before invoking **wsadmin** in interactive mode. The purpose of the script profile is to customize the environment in which script runs. For example, a script profile can be set for the Jacl scripting language that makes Jacl-specific variables or procedures available to the interactive session or main script.

The `-profile` command-line option can be used to specify a profile script. Several -profile options can be used on the command line and are invoked in the order given.

Connected versus local mode

The **wsadmin** command can operate in either connected or local mode. In connected mode, all operations are performed by method invocations on running JMX™ MBeans. In local mode, the application server (MBeans server) is not started and the **wsadmin** objects are limited to configuring the server by means of directly manipulating XML configuration documents.

When operating in local mode, be sure that you are operating on the correct profile, by either using the -profileName argument or starting wsadmin from the profile/bin directory.

When performing configuration changes in local mode in a distributed server environment, care should be take to make configuration changes at the deployment manager level. Changes made directly to the node configuration will be lost at server startup or at configuration replication.

Use the `-conntype NONE` option to run in local mode.

Command and script invocation

The **wsadmin** commands can be invoked in three different ways. This section describes how to invoke the command.

Note: For simplicity, the examples in this chapter will assume that:

- wsadmin is executed from the *profile_root*/bin directory, so it is not necessary to specify the profile name, host, and port.
- Administrative security is disabled. In reality, you will need to specify the username and password when you invoke wsadmin.

Invoking a single command (-c)

The -c option is used to execute a single command using **wsadmin** in Example 4. In the example, we use the AdminControl object to query the node name of the WebSphere server process.

Example 4   Running a single command in wsadmin

```
C:\WebSphereV7\AppServer\profiles\jmgr40\bin>wsadmin -lang jython -c
AdminControl.getNode()
WASX7209I: Connected to process "jobmgr" on node jmgr40node using SOAP
connector
;  The type of process is: JobManager
```

```
'jmgr40node'

C:\WebSphereV7\AppServer\profiles\jmgr40\bin>
```

Running script files (-f)

The -f option is used to execute a script file. Example 5 shows a two-line Jython script named myScript.py. The script has a .py extension to reflect the Jython language syntax of the script. The extension plays no significance in **wsadmin**; the com.ibm.ws.scripting.defaultLang property or -lang parameter is used to determine the language used. If the property setting is not correct, use the -lang option to identify the scripting language, because the default is Jacl.

Example 5 Jython script

```
print "This is an example Jython script"
print ""+ AdminControl.getNode()+""
```

Example 6 shows how to execute the script.

Example 6 Running a Jython script in wsadmin

```
C:\WebSphereV7\AppServer\profiles\dmgr40\bin>wsadmin -f myScript.py
-lang jython

WASX7209I: Connected to process "dmgr" on node dmgr40node using SOAP
connector;
 The type of process is: DeploymentManager
This is an example Jython script
dmgr40node
```

Invoking commands interactively

The command execution environment can be run in interactive mode, so you can invoke multiple commands without having the overhead of starting and stopping the **wsadmin** environment for every single command. Run the **wsadmin** command without the command (-c) or script file (-f) options to start the interactive command execution environment, as shown in Example 7.

Example 7 starting the wsadmin interactive command execution environment

```
C:\WebSphereV7\AppServer\profiles\dmgr40\bin>wsadmin -lang jython
WASX7209I: Connected to process "dmgr" on node dmgr40node using SOAP
connector;
 The type of process is: DeploymentManager
WASX7031I: For help, enter: "print Help.help()"
```

```
wsadmin>
```

From the `wsadmin>` prompt, the WebSphere administrative objects and built-in language objects can be invoked, as shown in Example 8. Simply type the commands at the wsadmin> prompt.

Example 8 interactive command invocation

```
wsadmin>AdminControl.getNode()
'dmgr40node'
wsadmin>
```

End the interactive execution environment by typing **`quit`** and pressing the Enter key.

wsadmin management objects

Five management objects are available for wsadmin, as shown in Figure 1. Prior to V7, the use of these objects in a wsadmin script could be quite complex and difficult to script. Command assist was available in the administrative console to provide the equivalent script commands for certain tasks performed in the console, but putting these commands to use in a script required some work.

New in V7: WebSphere Application Server V7.0 includes script libraries that can simplify the use of these objects. For that reason, in this chapter about the use of the management objects, refer to the Information Center. Note that the command assist feature has been expanded in the administrative console to include more tasks. Using the script libraries and command assist will be discussed more fully in later sections.

The wsadmin tool has the following five administrative objects that provide server configuration and management capabilities:

- Help
- AdminControl
- AdminConfig
- AdminApp
- AdminTask

Help

The Help object provides a quick way to get information about methods, operations, and attributes while using scripting.

For example, to get a list of the public methods available for the AdminControl object, enter the following command as shown:

```
wsadmin>print Help.AdminConfig()
```

To get a detailed description of a specific object method and the parameters it requires, invoke the help method of the target object with the method name as the option to the help method, as shown in Example 9.

Example 9 AdminConfig.help scripting

```
wsadmin>print AdminConfig.help("createClusterMember")
WASX7284I: Method: createClusterMember
        Arguments: cluster id, node id, member attributes

        Description: Creates a new Server object on the node specified
        by "node id."  This Server is created as a new member of the existing
        cluster specified by "cluster id," and has attributes specified in
        "member attributes."  One attribute is required: "memberName."
        The Server is created using the default template for
        Server objects, and has the name and specified by the
        "memberName" attribute.
        attribute.

        Method: createClusterMember

        Arguments: cluster id, node id, member attributes, template id

        Description: Creates a new Server object on the node specified
        by "node id."  This Server is created as a new member of the existing
        cluster specified by "cluster id," and has attributes specified in
        "member attributes."  One attribute is required: "memberName."
        The Server is created using the Server template
        specified by "template id," and has the name specified by
        the "memberName" attribute.
```

(New in V7) The AdminTask object supports a new help function that allows you to search for the specific command by using a wildcard character. For example, the command to get a list of the command names that start with “create” is shown in Example 10 on page 11.

Example 10 AdminTask.help scripting

```
wsadmin>print AdminTask.help("-commands", "create*")
WASX8004I: Available admin commands:

createAllActivePolicy - Create a policy that automatically activates all group
members.
createApplicationServer - Command that creates a server
createApplicationServerTemplate - creates a server Template based on a server
configuration
createAuditEncryptionConfig - Configures audit record encryption.
createAuditEventFactory - Creates an entry in the audit.xml to reference the
configuration of a
 Factory interface.
createAuditFilter - Creates an entry in the audit.xml to reference an Audit
Specification. Enab
createAuditKeyStore - Creates a new Key Store.
createAuditNotification - Configures an audit notification.
createAuditNotificationMonitor - Configures an audit notification monitor.
createAuditSelfSignedCertificate - Create a new self-signed certificate and store it
in a keys
....
```

AdminControl

The AdminControl object is used for operational control. It communicates with MBeans that represent live objects running a WebSphere server process. It includes commands to query existing running objects and their attributes and invoke operations on the objects. In addition to the operational commands, the AdminControl object supports commands to query information about the connected server, convenient commands for client tracing, reconnecting to a server, and starting and stopping a server.

Note that because the AdminControl object operates on live MBeans, it cannot be used to start a deployment manager, node agent, or standalone application server.

AdminConfig

The AdminConfig object is used to manage the configuration information that is stored in the repository. This object communicates with the WebSphere Application Server configuration service component to make configuration inquires and changes. You can use it to query existing configuration objects, create configuration objects, modify existing objects, and remove configuration objects. In a distributed server environment, the AdminConfig commands are available only if a scripting client is connected to the deployment manager. When connected to a node agent or a managed application server, the AdminConfig commands will not be available because the configuration for these server processes are copies of the master configuration that resides in the deployment manager.

AdminApp

The AdminApp object can update application metadata, map virtual hosts to Web modules, and map servers to modules for applications already installed. Changes to an application, such as specifying a library for the application to use or setting session management configuration properties, are performed using the AdminConfig object.

AdminTask

The AdminTask object is used to access a set of task-oriented administrative commands that provide an alternative way to access the configuration commands and the running object management commands. The administrative commands run simple and complex commands. The administrative commands are discovered dynamically when the scripting client is started. The set of available administrative commands depends on the edition of WebSphere Application Server you install. You can use the AdminTask object commands to access these commands.

Two run modes are always available for each administrative command, namely the batch and interactive mode. When you use an administrative command in interactive mode, you go through a series of steps to collect your input interactively. This process provides users a text-based wizard and a similar user experience to the wizard in the administrative console. You can also use the help command to obtain help for any of the administrative commands and the AdminTask object.

New functions have been added to the AdminTask object to support V7 features, such as the certificate authority, business level applications, proxy management, and to support the use of properties file based configurations.

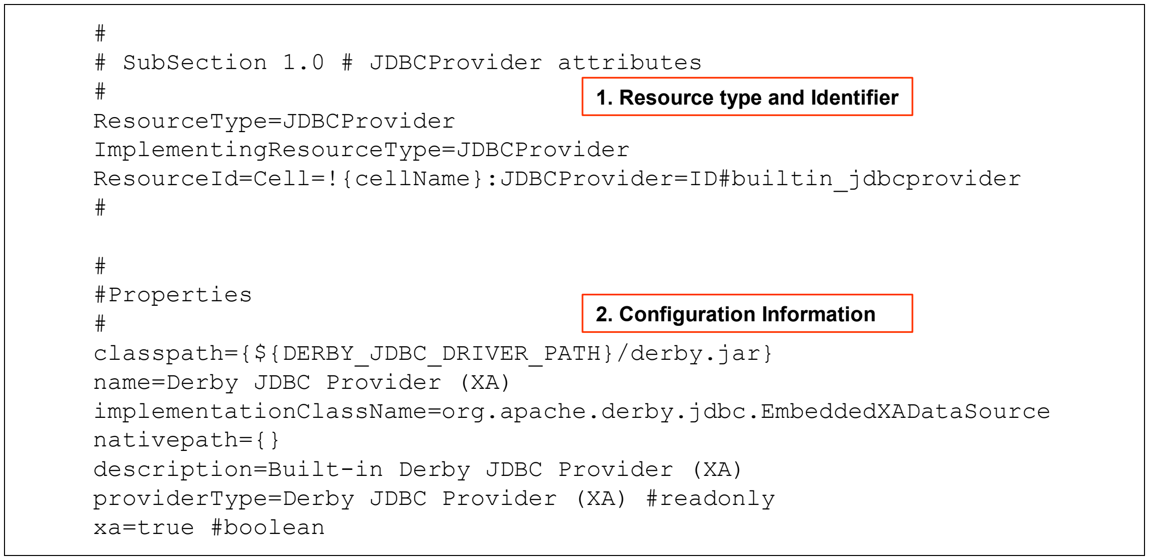
(New in V7) Properties file based configuration

Using complex scripts requires knowledge of Jacl or Jython scripting language, as well as the public MBean interfaces. The use of properties file based configuration provides a way to simplify administrative tasks using wsadmin.

To implement this type of configuration, five new commands have been added to the AdminTask object in the PropertiesBasedConfiguration command group:

- extractConfigProperties
- validateConfigProperties
- applyConfigProperties
- deleteConfigProperties
- createPropertiesFileTemplates

Properties file based configuration extracts configuration data to one file that is easy to read using any editor tool. The configuration attributes are provided as key/value pairs (Figure 2).


1. Resource type and Identifier
2. Configuration Information

```
#
# SubSection 1.0 # JDBCProvider attributes
#
ResourceType=JDBCProvider
ImplementingResourceType=JDBCProvider
ResourceId=Cell=!{cellName}:JDBCProvider=ID#builtin_jdbcprovider
#

#
#Properties
#
classpath={${DERBY_JDBC_DRIVER_PATH}/derby.jar}
name=Derby JDBC Provider (XA)
implementationClassName=org.apache.derby.jdbc.EmbeddedXADataSource
nativepath={}
description=Built-in Derby JDBC Provider (XA)
providerType=Derby JDBC Provider (XA) #readonly
xa=true #boolean
```

Figure 2 Example: properties for a JDBC™ provider

After the properties have been extracted, you make any necessary changes to the attributes and validate the changes before applying them to the server. It is also possible to create or delete configuration objects.

Samples of these commands are shown in Example 11.

Example 11 wsadmin commands for properties based configuration

```
AdminTask.extractConfigProperties('-configData Node=myNode
-propertiesFileName myNodeProperties.props')

AdminTask.validateConfigProperties('-propertiesFileName
myNodeProperties.props -reportFile report.txt')
```

```
AdminTask.applyConfigProperties('-propertiesFileName myPropFile.props
-validate true')

AdminTask.deleteConfigProperties('-propertiesFileName
myPropFile.props')

AdminTask.createPropertiesFileTemplates('-propertiesFileName
serverTemplate.props -configType Server')
```

Because it is not possible to modify every configuration using properties file based configuration, we do *not* recommend that you use this tool for backup and recovery. BackupConfig and RestoreConfig found in the *<was_home>*/bin directory should still be used as the main backup and recovery tool.

Managing WebSphere using script libraries

(New in V7) Script libraries can be used to perform a higher level of wsadmin functions than can be done using a single wsadmin command. Only a single line from a library function is needed to perform complex functions. Each script is written in Jython, and is often referred to as “the Jython script”. The script libraries are categorized into six types and the types are subdivided (see Table 2).

Python script files are supplied for each Jython class file. The Python files can be read as a text files. Table 2 shows the scripts and the type of resource they manage. Each script has a set of procedures that perform specific functions.

Table 2 The types of the script library

TYPE	***Python (Jython) script file***
Application	AdminApplication AdminBLA
Resources	AdminJ2C AdminJDBC AdminJMS AdminResources
Security	AdminAuthorizations
Servers	AdminClusterManagement AdminServerManagement
System	AdminNodeGroupManagement AdminNodeManagement

TYPE	*Python (Jython) script file*
Application	AdminApplication AdminBLA
Utilities	AdminLibHelp AdminUtilities

Script libraries, their procedures, and syntax: More information about these script libraries can be found in the WebSphere Application Server Information Center as sub-topics under:

- Jython script files:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/welc_ref_adm_jython.html`

Invoking script libraries

The script libraries are located in *install_root* /scriptLibraries directory. These libraries are loaded when wsadmin starts and are readily available from the wsadmin command prompt or to be used from the customized scripts. You can invoke the scripts in interactive mode or script mode.

Interactive mode

Interactive mode is suitable for simple tasks and testing. Using this mode allows you to get the results directly. To invoke a script interactively, start a wsadmin session and enter the script name, procedure, and arguments at the wsadmin> prompt (see Example 12).

Example 12   Using the Jython scripts in interactive mode

```
C:\WebSphere\AppServer\bin>wsadmin -lang jython
WASX7209I: Connected to process "dmgr" on node DmgrNode02 using SOAP
connector;
 The type of process is: DeploymentManager
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminServerManagement.checkIfServerExists("sys1Node01",
"Amember01")
---------------------------------------------------------------
 AdminServerManagement:  Check if server exists
 Node name:              sys1Node01
 Server name:            Amember01
 Usage: AdminServerManagement.checkIfServerExists("sys1Node01",
"Amember01")
---------------------------------------------------------------

'true'
wsadmin>
```

Script file mode (-f)

Using a script file with wsadmin is useful when you want to have daily tasks performed automatically or if you need to manage multiple servers.

To run in script mode, you select the script libraries to use and merge them into your own script file. Save the custom script as a Python file and run it from the command line.

Example 13 shows a Python file containing two script library commands.

Example 13 test.py script

```
# Writting by Jython
# Script file name : test.py

AdminServerManagement.checkIfServerExists("sys1Node01", "Amember21")
AdminServerManagement.createApplicationServer(“sys1Node01”,
“Amember21”)
```

Example 14 shows how to invoke the script.

Example 14

```
C:\WebSphere\AppServer\bin>
C:\WebSphere\AppServer\bin>wsadmin.bat -lang jython -f C:\temp\test.py
```

Customizing scripts

This is an advanced use of the script mode. You can add customized code written in Python or Jython to your script file (the one that calls the script libraries).

Note: *DO NOT* modify the script libraries. If you need to customize the scripts, you can copy parts of the library code to other files and modify the copied code to improve it or to better suit your needs.

When customizing a script, it is best to do this in three steps:

1. Run each Jython script in interactive mode, verifying the syntax and the result.
2. Create the script file that will call the script libraries, combining all Jython scripts. Verify that the results are as you expect.
3. Add your additional **wsadmin** commands, written in Python, and verify that the customized script does the work that you intended.

Displaying help for script libraries

You can use AdminLibHelp script to display each script within a script library. For example, the following command invocation displays each script in the AdminApplication script library:

```
print AdminLibHelp.help("AdminApplication")
```

You can use the help script to display detailed descriptions, arguments, and usage information for a specific script. For example, the following command invocation displays detailed script information for the listApplications script in the AdminApplication script library:

```
print AdminApplication.help('listApplications')
```

Example 15 shows sample code for displaying help information for the createApplicationServer procedure in the AdminServerManagement script.

Example 15 Help information for a procedure in createApplicationServer

```
wsadmin>print AdminServerManagement.help('createApplicationServer')
WASL2004I: Procedure: createApplicationServer

        Arguments: nodeName, serverName, (Optional) templateName

        Description: Create a new application server

       Usage: AdminServerManagement.createApplicationServer( nodeName,
serverName, templateName)
wsadmin>
```

Application script library

The application scripts provide a set of procedures to manage and configure enterprise applications and business level applications. These scripts can be used individually, or combined in a custom script file to automate application installation, configuration, and management tasks.

The library located at *install_root/*scriptlibraries/application/V70 directory, contains two scripts:

- AdminApplication, which provides procedures to manage enterprise applications
- AdminBLA, which provides procedures to manage business level applications

The AdminApplication script is discussed in the next section. For information about the AdminBLA script, see:

- Business-level application configuration scripts

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/rxml_7libbla.html`

AdminApplication script

The AdminApplication script contains procedures that allow you to manage enterprise applications.

The syntax of the AdminApplication script is:

`AdminApplication.`*`procedure_name`*`(`*`arguments`*`)`

The following list summarizes the script procedures available with this script. The name of the procedure gives you a good idea of the function it provides.

- Install and uninstall applications:
  - installAppWithAppNameOption
  - installAppWithDefaultBindingOption
  - installAppWithNodeAndServerOptions
  - installAppWithClusterOption
  - installAppModulesToSameServerWithMapModulesToServersOption
  - installAppModulesToDiffServersWithMapModulesToServersOption
  - installAppModulesToSameServerWithPatternMatching
  - installAppModulesToDiffServersWithPatternMatching
  - installAppModulesToMultiServersWithPatternMatching
  - installAppWithTargetOption
  - installAppWithDeployEjbOptions
  - installAppWithVariousTasksAndNonTasksOptions
  - installWarFile
  - uninstallApplication
- Update applications:
  - addSingleFileToAnAppWithUpdateCommand

  - addSingleModuleFileToAnAppWithUpdateCommand
  - addUpdateSingleModuleFileToAnAppWithUpdateCommand
  - addPartialAppToAnAppWithUpdateCommand
  - deleteSingleFileToAnAppWithUpdateCommand
  - deleteSingleModuleFileToAnAppWithUpdateCommand
  - deletePartialAppToAnAppWithUpdateCommand
  - updateApplicationUsingDefaultMerge
  - updateApplicationWithUpdateIgnoreNewOption
  - updateApplicationWithUpdateIgnoreOldOption
  - updateEntireAppToAnAppWithUpdateCommand
  - updatePartialAppToAnAppWithUpdateCommand
  - updateSingleFileToAnAppWithUpdateCommand
  - updateSingleModuleFileToAnAppWithUpdateCommand
- Export applications:
  - exportAnAppToFile
  - exportAllApplicationsToDir
  - exportAnAppDDLToDir
- Configure application deployment characteristics:
  - configureStartingWeightForAnApplication
  - configureClassLoaderPolicyForAnApplication
  - configureClassLoaderLoadingModeForAnApplication
  - configureSessionManagementForAnApplication
  - configureApplicationLoading
  - configureLibraryReferenceForAnApplication
  - configureEJBModulesOfAnApplication
  - configureWebModulesOfAnApplication
  - configureConnectorModulesOfAnApplication
- Start and stop enterprise and business level applications:
  - startApplicationOnSingleServer
  - startApplicationOnAllDeployedTargets
  - startApplicationOnCluster
  - stopApplicationOnSingleServer
  - stopApplicationOnAllDeployedTargets
  - stopApplicationOnCluster
- Query applications:
  - checkIfAppExists
  - getAppDeployedNodes
  - getAppDeploymentTarget
  - getTaskInfoForAnApp
  - listApplications
  - listApplicationsWithTarget

– listModulesInAnApp

Example: Installing an application

There are multiple procedures that can be used to install applications. When preparing to install an application, determine the options you will need and choose the procedure accordingly.

The most basic procedure is installAppWithAppwithNodeAndServerOptions. If the installation is successful, the message returns, `installed application successfully`.

The syntax to invoke this procedure is:

AdminApplication.installAppWithNodeAndServerOptions(*app_name, app_location, node_name, app_server_Name*)

Example 16 illustrates this procedure.

Example 16 Install application script library

```
wsadmin>AdminApplication.installAppWithNodeAndServerOptions("IBMUTC", 
"C:/IBMUTC.ear", "sys1Node01", "Amember01")
---------------------------------------------------------------
 AdminApplication:       Install application with -node and -server 
options
 Application name:       IBMUTC
 Ear file to deploy:     C:/IBMUTC.ear
 Node name:              sys1Node01
 Server name:            Amember01
 Usage: AdminApplication.installAppWithNodeAndServerOptions("IBMUTC", 
"C:/IBMUTC
.ear", "sys1Node01", "Amember01")
---------------------------------------------------------------
WASX7327I: Contents of was.policy file:grant codeBase 
"file:${application}" {permission java.security.AllPermission;
};
ADMA5016I: Installation of IBMUTC started.
ADMA5058I: Application and module versions are validated with versions 
of deploy
ment targets.

CWSAD0040I: The application IBMUTC is configured in the Application 
Server repos
itory.
ADMA5113I: Activation plan created successfully.
```

```
ADMA5011I: The cleanup of the temp directory for application IBMUTC is
complete.

ADMA5013I: Application IBMUTC installed successfully.
OK: installAppWithNodeAndServerOptions('IBMUTC', 'C:/IBMUTC.ear',
'sys1Node01',
'Amember01', 'false'):
```

Example: Starting an application

Multple procedures are also available to start an application. As start the application, you should choose which script is the most suitable.

The startApplicationSingleServer procedure will start a single application on a single application server. The syntax is:

`AdminApplication.startApplicationOnSingleServer(`*`app_name, node_name, app_server_name)`*

Example 17 illustrates this procedure.

Example 17 start application script library

```
wsadmin>AdminApplication.startApplicationOnSingleServer("IBMUTC","sys1N
ode01","Amember01")
---------------------------------------------------------------
 AdminApplication:       Start an application on a single server
 Application name:       IBMUTC
 Node name:              sys1Node01
 Server name:            Amember01
 Usage: AdminApplication.startApplicationOnSingleServer("IBMUTC",
"sys1Node01","Amember01")
---------------------------------------------------------------
OK: startApplicationOnSingleServer('IBMUTC', 'sys1Node01', 'Amember01',
'false')
:1
```

Resource script library

The Resource script library provides a set of scripts to manage WebSphere resources. The library provides script functions for J2C resources, JDBC providers, and JMS resources at the server scope. If you need to configure resources at the cell, node, or cluster level, it is possible to customize the scripts for this purpose.

The script library is located *install_root/*scriptlibraries/resources/V70 directory. It contains the following scripts and procedures:

- AdminJ2C script:

  The following procedures allow you to configure J2C resources:

  – createJ2CActivationSpec
  – createJ2CAdminObject
  – createJ2CConnectionFactory
  – installJ2CResourceAdapter

  The following procedures allow you to query J2C resources:

  – listAdminObjectInterfaces
  – listConnectionFactoryInterfaces
  – listJ2CActivationSpecs
  – listJ2CAdminObjects
  – listJ2CConnectionFactories
  – listJ2CResourceAdapters
  – listMessageListenerTypes

- AdminJDBC script:

  The following procedures allow you to configure JDBC resources:

  – createDataSource
  – createDataSourceUsingTemplate
  – createJDBCProvider
  – createJDBCProviderUsingTemplate

  The following procedures allow you to query JDBC resources:

  – listDataSources
  – listDataSourceTemplates
  – listJDBCProviders
  – listJDBCProviderTemplates

- AdminJMS script:

  The following procedures allow you to configure JMS resources:

  – createGenericJMSConnectionFactory
  – createGenericJMSConnectionFactoryUsingTemplate
  – createGenericJMSDestination
  – createGenericJMSDestinationUsingTemplate
  – createJMSProvider
  – createJMSProviderUsingTemplate
  – createWASQueue
  – createWASQueueUsingTemplate
  – createWASQueueConnectionFactory
  – createWASQueueConnectionFactoryUsingTemplate

  - createWASTopic
  - createWASTopicUsingTemplate
  - createWASTopicConnectionFactory
  - createWASTopicConnectionFactoryUsingTemplate
  - startListenerPort

  The following procedures allow you to query JMS resources:

  - listGenericJMSConnectionFactories
  - listGenericJMSConnectionFactoryTemplates
  - listGenericJMSDestinations
  - listGenericJMSDestinationTemplates
  - listJMSProviders
  - listJMSProviderTemplates
  - listWASQueueConnectionFactoryTemplates
  - listWASQueueTemplates
  - listWASTopicConnectionFactoryTemplates
  - listWASQueueConnectionFactories
  - listWASQueues
  - listWASTopicConnectionFactories
  - listWASTopics
  - listWASTopicTemplates

- AdminResources:

  Use the following script procedures to configure your mail settings:

  - createCompleteMailProvider
  - createMailProvider
  - createMailSession
  - createProtocolProvider

  Use the following script procedures to configure your resource environment settings:

  - createCompleteResourceEnvProvider
  - createResourceEnvEntries
  - createResourceEnvProvider
  - createResourceEnvProviderRef

  Use the following script procedures to configure your URL provider settings:

  - createCompleteURLProvider
  - createURL

  Use the following script procedures to configure additional Java Enterprise Edition (JEE) resources:

  - createJAASAuthenticationAlias
  - createLibraryRef
  - createSharedLibrary

– createScheduler
– createWorkManager

Example: Listing JDBC resources

The listDataSources and listJDBCProviders procedures of the AdminJDBC script can be used to display a list of configuration IDs for the JDBC providers and data sources in your environment.

The syntax to use each procedure is:

- `AdminJDBC.listDataSources(`*`ds_name`*`)`
- `AdminJDBC.listJDBCProviders(`*`jdbc_name`*`)`

Example 18 shows the use of these procedures.

Example 18 list up JDBC resources

```
wsadmin>AdminJDBC.listDataSources("PLANTSDB")
---------------------------------------------------------------
 AdminJDBC:              listDataSources
 Optional parameter:
 DataSource name:        PLANTSDB
 Usage: AdminJDBC.listDataSources("PLANTSDB")
---------------------------------------------------------------
['PLANTSDB(cells/Cell02/nodes/sys1Node01/servers/server1|resources.xml#
DataSource_1183122165968)']
wsadmin>
wsadmin>AdminJDBC.listJDBCProviders("Derby JDBC Provider")
---------------------------------------------------------------
 AdminJDBC:               listJDBCProviders
 Optional parameter:
 JDBC provider name:     Derby JDBC Provider
 Usage: AdminJDBC.listJDBCProvider("Derby JDBC Provider")
---------------------------------------------------------------

['"Derby JDBC
Provider(cells/Cell02/nodes/sys1Node01/servers/server1|resources.x
ml#JDBCProvider_1183122153343)"']
```

Example: Creating a J2C connection factory

The createJ2CConnectionFactory procedure in the AdminJ2C script creates a new J2C connection factory in the environment. The result is the configuration ID of the new J2C connection factory.

The arguments are the resource adapter, connection factory name, the connection factory interface, and the Java Naming and Directory Interface (JNDI) name arguments. The syntax is:

AdminJ2C.createJ2CConnectionFactory(*resource_adapterID, connfactory_name, connFactory_interface, jndi_name, attributes*)

Example 19 shows sample coding for creating a J2C connection factory.

Example 19   createJ2CConnectionFactory

```
wsadmin>AdminJ2C.createJ2CActivationSpec("WebSphere MQ Resource
Adapter(cells/Cell02/nodes/DmgrNode02/servers/dmgr|resources.xml#J2CRes
ourceAdapter_1234298429000)", "WebSphere MQ Resource Adapter",
"javax.jms.MessageListener", "jdbc/PlantsByWebSphereDataSourceNONJTA")
---------------------------------------------------------------
 AdminJ2C:                        createJ2CActivationSpec
 J2CResourceAdapter configID:     WebSphere MQ Resource
Adapter(cells/Cell02/nodes/DmgrNode02/servers/dmgr|resources.xml#J2CRes
ourceAdapter_1234298429000)
 J2CActivationSpec name:          WebSphere MQ Resource Adapter
 Message listener type:           javax.jms.MessageListener
 jndi name:                       jdbc/PlantsByWebSphereDataSourceNONJTA
 Optional attributes:
    otherAttributesList  []
 Usage: AdminJ2C.createJ2CActivationSpec("WebSphere MQ Resource
Adapter(cells/Cell02/nodes/DmgrNode02/servers/dmgr|resources.xml#J2CRes
ourceAdapter_1234298429000)", "WebSphere MQ Resource Adapter",
"javax.jms.MessageListener", "jdbc/PlantsB
yWebSphereDataSourceNONJTA")
---------------------------------------------------------------

'"WebSphere MQ Resource
Adapter(cells/Cell02/nodes/DmgrNode02/servers/dmgr|resou
rces.xml#J2CActivationSpec_1236206121468)"'
wsadmin>
```

Security script library

The security script library provides a script to manage the security. This library is located *install_root*/scriptlibraries/security/V70 directory.

The AdminAuthorizations script has the following procedures:

- Configure authorization groups:
  - addResourceToAuthorizationGroup
  - createAuthorizationGroup
  - mapGroupsToAdminRole
  - mapUsersToAdminRole
- Remove users and groups from the security authorization settings:
  - deleteAuthorizationGroup
  - removeGroupFromAllAdminRoles
  - removeGroupsFromAdminRole
  - removeResourceFromAuthorizationGroup
  - removeUserFromAllAdminRoles
  - removeUsersFromAdminRole
- Query your security authorization group configuration:
  - listAuthorizationGroups
  - listAuthorizationGroupsForUserID
  - listAuthorizationGroupsForGroupID
  - listAuthorizationGroupsOfResource
  - listUserIDsOfAuthorizationGroup
  - listGroupIDsOfAuthorizationGroup
  - listResourcesOfAuthorizationGroup
  - listResourcesForUserID

Example: Listing the authorization groups

The listAuthorizationGroups procedure displays each authorization group in the security configuration. This script does not require arguments.

```
AdminAuthorizations.listAuthorizationGroups()
```

Example 20 shows sample coding for listing the authorization groups.

Example 20 Listing authorization groups

```
wsadmin>AdminAuthorizations.listAuthorizationGroups()
---------------------------------------------------------------
 AdminAuthorizations:  List authorization groups
 Usage: AdminAuthorizations.listAuthorizationGroups()
---------------------------------------------------------------

['sec_group1']
```

Server script library

The server script library provides a set of scripts and their procedures to manage the server and the cluster component.

This library is located *install_root*/scriptlibraries/servers/V70 directory. The library contains the following scripts and procedures:

- AdminServerManagement script:

  Start and stop servers:

  – startAllServers
  – startSingleServer
  – stopAllServers
  – stopSingleServer

  Configure servers:

  – createApplicationServer
  – createAppServerTemplate
  – createGenericServer
  – createWebServer
  – deleteServer
  – deleteServerTemplate

  Query the server configuration:

  – checkIfServerExists
  – checkIfServerTemplateExists
  – getJavaHome
  – getServerProcessType
  – getServerPID
  – help
  – listJVMProperties
  – listServers
  – listServerTemplates
  – listServerTypes
  – queryingMBeans
  – showServerInfo
  – viewingProductInformation

  Manage server settings:

  – configureAdminService
  – configureApplicationServerClassloader
  – configureDynamicCache
  – configureEJBContainer
  – configureFileTransferService
  – configureListenerPortForMessageListenerService

  - configureMessageListenerService
  - configureStateManageable
  - configureCustomProperty
  - configureCustomService
  - configureEndPointsHost
  - configureJavaVirtualMachine
  - configureORBService
  - configureProcessDefinition
  - configureRuntimeTransactionService
  - configureThreadPool
  - configureTransactionService
  - setJVMProperties
  - setTraceSpecification
  - configureCookieForServer
  - configureHTTPTransportForWebContainer
  - configureSessionManagerForServer
  - configureWebContainer
  - configureJavaProcessLogs
  - configurePerformanceMonitoringService
  - configurePMIRequestMetrics
  - configureRASLoggingService
  - configureServerLogs
  - configureTraceService

- AdminClusterManagement script

  Start cluster processes:

  - rippleStartAllClusters
  - rippleStartSingleCluster
  - startAllClusters
  - startSingleCluster

  Stop cluster processes:

  - immediateStopAllRunningClusters
  - immediateStopSingleCluster
  - stopAllClusters
  - stopSingleCluster

  Configure clusters:

  - createClusterMember
  - createClusterWithFirstMember
  - createClusterWithoutMember
  - createFirstClusterMemberWithTemplate
  - createFirstClusterMemberWithTemplateNodeServer

  Remove clusters and cluster members:

  - deleteCluster
  - deleteClusterMember

  Query a cluster configuration:

  - checkIfClusterExists
  - checkIfClusterMemberExists
  - help
  - listClusters
  - listClusterMembers

Example: Creating an application server

The CreateApplicationServer procedure of the AdminServerManagement script creates a new application server. The script requires the node to be running. The syntax is:

`AdminServerManagement.createApplicationServer(`*`node_name, server_name,Template`*`)`

Example 21 illustrates sample coding for creating an application server.

Example 21 Creating an application server

```
wsadmin>AdminServerManagement.createApplicationServer("sys1Node01","Ame
mber01","default")
---------------------------------------------------------------
 AdminServerManagement:  Create an application server on a given node
 Node name:              sys1Node01
 New Server name:        Amember01
 Optional parameter:
Template name:  default
 Usage: AdminServerManagement.createApplicationServer("sys1Node01",
"Amember01", "default")
---------------------------------------------------------------
'Amember01(cells/Cell02/nodes/sys1Node01/servers/Amember01|server.xml#Server_12
35061945890)'
```

Example: Starting an application server

The StartAllServers procedure of the AminServerManagement script starts all application servers on a node. StartSingleServer starts one server. The syntax for these two procedures is:

- `AdminServerManagement.startAllServers(`*`node_name)`*
- `AdminServerManagement.startSingleSever(`*`node_name, server_name)`*

Example 22 shows sample coding for starting an application server.

Example 22 Starting the application server using a single script library

```
wsadmin>AdminServerManagement.startAllServers("sys1Node01")
---------------------------------------------------------------
 AdminServerManagement:   Start all servers
 Node name:               sys1Node01
 Usage: AdminServerManagement.startAllServers("sys1Node01")
---------------------------------------------------------------
Start server: Amember01
Start server: server1
OK: startAllServers('sys1Node01', 'false'):1

wsadmin>AdminServerManagement.startSingleServer("sys1Node01", 
"Amember01")
---------------------------------------------------------------
 AdminServerManagement:  Start single server
 Node name:              sys1Node01
 Server name:            Amember01
 Usage: AdminServerManagement.startSingleServer("sys1Node01", 
"Amember01")
---------------------------------------------------------------
Start server: Amember01
OK: startSingleServer('sys1Node01', 'Amember01', 'false'):1
```

Example: Stopping application servers

The StopAllServers procedure stops all application servers on a node. The StopSingleServer will stop one server. The syntax for these procedures is:

- `AdminServerManagement.stopAllServers(`*`node_name)`*
- `AdminServerManagement.stopSingleSever(`*`node_name, server_name)`*

Example 23 shows sample coding for stopping application servers.

Example 23 Stopping the application server using a single script library

```
wsadmin>AdminServerManagement.stopAllServers("sys1Node01")
---------------------------------------------------------------
 AdminServerManagement:   Stop all servers
 Node name:               sys1Node01
 Usage: AdminServerManagement.stopAllServers("sys1Node01")
---------------------------------------------------------------
Stop server: Amember01
WASX7337I: Invoked stop for server "Amember01" Waiting for stop 
completion.
Stop server: server1
```

```
WASX7337I: Invoked stop for server "server1" Waiting for stop
completion.
OK: stopAllServers('sys1Node01', 'false'):1

wsadmin>AdminServerManagement.stopSingleServer("sys1Node01",
"Amember01")
---------------------------------------------------------------
 AdminServerManagement:  Stop single server
 Node name:              sys1Node01
 Server name:            Amember01
 Usage: AdminServerManagement.stopSingleServer("sys1Node01",
"Amember01")
---------------------------------------------------------------
Stop server: Amember01
WASX7337I: Invoked stop for server "Amember01" Waiting for stop
completion.
OK: stopSingleServer('sys1Node01', 'Amember01', 'false'):1
```

System management script library

The system management script library provides a set of scripts that manage nodes and node groups.

This library is located at *install_root*/scriptlibraries/system/V70 directory. It contains the following scripts and procedures:

- AdminNodeManagement
  - configureDiscoveryProtocolOnNode
  - doesNodeExist
  - isNodeRunning
  - listNodes
  - restartActiveNodes
  - restartNodeAgent
  - stopNode
  - stopNodeAgent
  - syncActiveNodes
  - syncNode
- AdminNodeGroupManagement
  - addNodeGroupMember
  - checkIfNodeExists
  - checkIfNodeGroupExists
  - createNodeGroup
  - createNodeGroupProperty

- deleteNodeGroup
- deleteNodeGroupMember
- deleteNodeGroupProperty
- help
- listNodeGroups
- listNodeGroupMembers
- listNodeGroupProperties
- modifyNodeGroup
- modifyNodeGroupProperty

Example: Querying node group members

The listNodeGroupMembers procedure lists the name of each node that is configured within a specific node group. The syntax is:

```
AdminNodeGroupManagement.listNodeGroupMembers(node_group_name)
```

Example 24 shows sample coding for querying node group members.

Example 24 listing node group members using script library

```
wsadmin>AdminNodeGroupManagement.listNodeGroupMembers("DefaultNodeGroup
")
---------------------------------------------------------------
 AdminNodeGroupManagement:       List nodes for a given node group
 Optional parameter:
         Node group name:        DefaultNodeGroup
 Usage:
AdminNodeGroupManagement.listNodeGroupMembers("DefaultNodeGroup")
---------------------------------------------------------------

['DmgrNode02', 'sys1Node01']
```

Example: Synchronizing a node

The synActiveNodes and syncNode procedures propagate a configuration change. The syntax for these commands is:

- `AdminNodeManagement.syncActiveNodes()`
- `AdminNodeManagement.syncNode(node_name)`

Example 25 shows sample coding for synchronizing a node.

Example 25 sync the node using scriptlibrary

```
wsadmin>AdminNodeManagement.syncNode("sys1Node01")
---------------------------------------------------------------
 AdminNodeManagement:          syncNode
 nodeName:                      sys1Node01
 Usage: AdminNodeManagement.syncNode("sys1Node01")
---------------------------------------------------------------
true
1
```

Assistance with scripting

When you perform an action in the administrative console, you can use the command assistance feature to show the corresponding scripting commands. This allows you to capture and copy them for use in wsadmin scripts. You also have the option to send these as notifications to the Rational® Application Developer V7.5, where you can use the Jython editor to build scripts.

Enabling command assistance

The command assistance feature in the administrative console was introduced in WebSphere Application Server V6.1 with limited scope in function. The command assistance feature has been broadened in V7.0.

When you perform an action in the administrative console, you can select the **View administrative scripting command for last action** option in the Help area of the panel to display the command equivalent. This command can be copied and pasted into a script or command window.

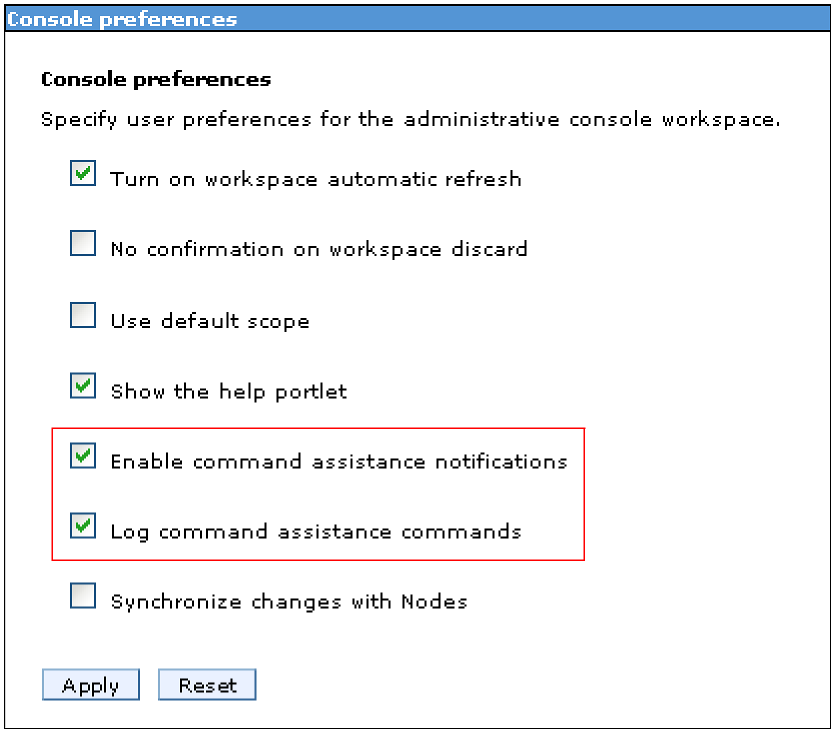
Two additional features can be enabled to assist in developing scripts.

To enable these features:

1. Go to **System administration** → **Console Preferences**.
2. Select the command assistance features you want to use (see Figure 3 on page 34):
   - Enable command assistance notifications:

     This option should be used in non-production environments to send notifications containing command assist data. Enablement of the notifications allows integration with Rational Application Developer.
   - Log command assistance commands:

     This option sends the commands to a log.


Console preferences

Console preferences

Specify user preferences for the administrative console workspace.

- [✓] Turn on workspace automatic refresh
- [ ] No confirmation on workspace discard
- [ ] Use default scope
- [✓] Show the help portlet
- [✓] Enable command assistance notifications
- [✓] Log command assistance commands
- [ ] Synchronize changes with Nodes

Figure 3 Administrative scripting command: mapping to actions

3. Click apply.

When you select the option to log commands, they are stored in the following location:

`profile_root/logs/AssistanceJythonCommands_user_name.log`

See Example 26 for some sample coding.

Example 26 log location

```
C:\WebSphere\AppServer\profiles\Dmgr01\logs\dmgr
\commandAssistanceJythonCommands_wasadmin.log
```

The first line of each log entry consists of a timestamp and the location within the console where the command was generated. Below the timestamp, is the command information. Example 27 on page 35 shows a sample of the log.

Example 27 the command assistance: log content

```
# [2/24/09 12:15:42:890 EST] Business-level applications > New
AdminTask.createEmptyBLA('[-name sample -description Sample ]')

# [2/24/09 12:15:42:906 EST] Business-level applications > New
AdminTask.listBLAs('[-blaID WebSphere:blaname=sample -includeDescription true
]')

# [2/24/09 12:15:42:906 EST] Business-level applications > New
AdminTask.listCompUnits('[-blaID WebSphere:blaname=sample -includeDescription
true -includeType true ]')

# [2/24/09 12:15:47:500 EST] Business-level applications > sample
AdminTask.listAssets('[-includeDescription true ]')
# [2/24/09 12:15:47:531 EST] Business-level applications > sample
AdminTask.listBLAs('[-includeDescription true ]')

# [2/24/09 12:15:50:531 EST] Business-level applications > sample > Add
AdminTask.addCompUnit('[-blaID WebSphere:blaname=sample -cuSourceID
WebSphere:blaname=IBMUTC ]')

# [2/24/09 12:15:53:562 EST] Business-level applications > sample > Add
AdminTask.addCompUnit('[-blaID WebSphere:blaname=sample -cuSourceID
WebSphere:blaname=IBMUTC -CUOptions [[WebSphere:blaname=sample
WebSphere:blaname=IBMUTC IBMUTC_0001 "" 1]]]')

# [2/24/09 12:15:57:625 EST] Adding composition unit to the business-level
application
AdminConfig.save()

# [2/24/09 12:15:57:890 EST] BLAManagement
AdminTask.listBLAs('[-includeDescription true ]')

# [2/24/09 12:16:01:421 EST] Business-level applications
AdminTask.startBLA('[-blaID WebSphere:blaname=sample ]')
```

Building script files using command assist

The command assist features provide several methods to build scripts. Commands can be copied from the Help area of the console, or from the log into Jython scripts.

The command assist notifications also provide an integration point with Rational Application Developer, which provides tools that allow you to monitor commands as they are created and to insert the monitored commands into a script.

Working with Jython scripts

To work with Jython scripts in Rational Application Developer, you create a Jython project and Jython script files in the project. This can be done in any perspective. When you open a new Jython script, it opens with the Jython editor.

The Jython editor in Rational Application Developer V7.5 is used to perform a variety of tasks, such as these:

- View the administrative console.
- Develop and edit Jython script files.
- Import existing Jython files for structured viewing.
- Set breakpoints for debugging your scripts.

The Jython editor has many text editing features, such as syntax highlighting, unlimited undo or redo, and automatic tab indentation.

When you tag a comment in a Jython script with "#TODO", the editor automatically creates a corresponding task as a reminder in the Tasks view. Then, if you open the task later, the editor automatically synchronizes to that TODO entry in the script source.

Other helpful features are content assist and tips, which provides a list of acceptable continuations depending on where the cursor is located in a Jython script file, or what you have just typed. The Jython editor is not integrated to a compiler. As a result, the Jython editor does not perform syntax verification on your scripts.

Using the command assist notifications

The command assistance in the administrative console sends JMX notifications containing command data. These notifications can be monitored from Rational Application Developer. This requires that you define the server producing the notifications as a server in the workspace.

To monitor the commands produced as actions are taken on the administrative console of the server, follow these steps:

1. In the Servers view, right-click the server and select **Administration** → **WebSphere administration command assist**. The WebSphere Administration Command view opens.
2. In the **Select Server to Monitor list**, specify the servers with a check mark that you want the tool to monitor as you interact with its administrative console. The Select Server to Monitor list is available in the toolbar of the WebSphere Administration Command view (see Figure 4).

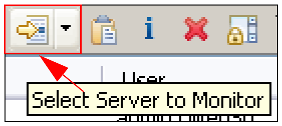
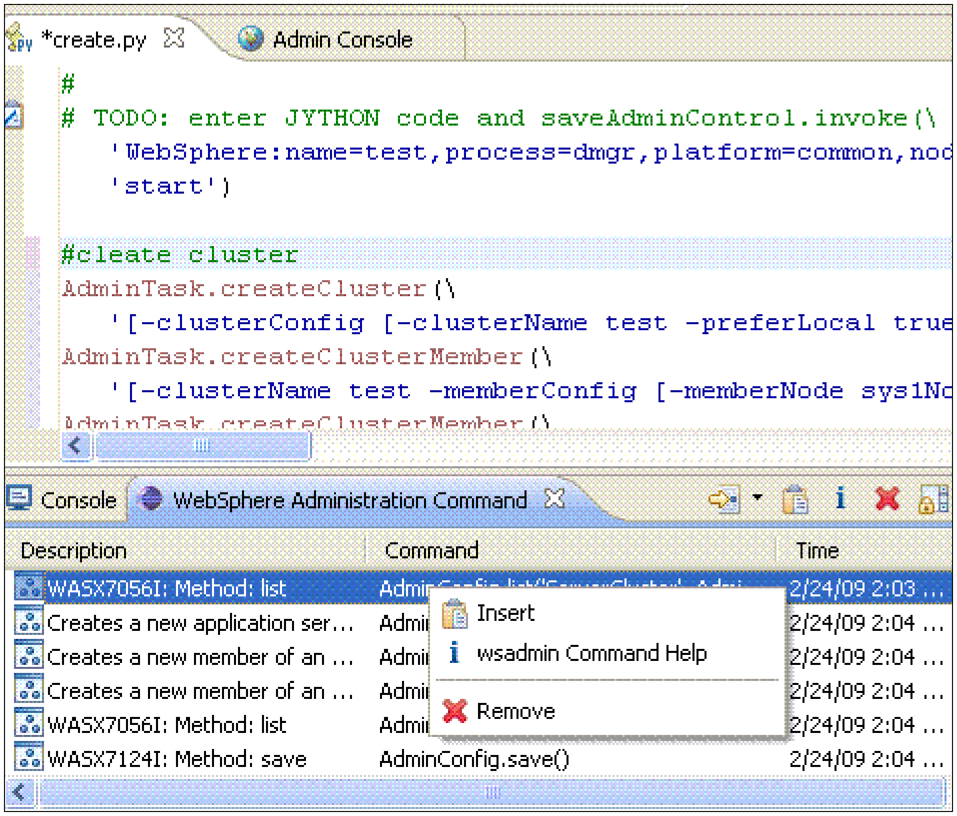


Figure 4 Select Server to Monitor icon

The server that you want to monitor needs to be started, started in profile or debug mode. The server is disabled for selection in the Select Server to Monitor list when the server is stopped.

As commands are generated by the console, they appear in the WebSphere Application Command Assist view. The commands are shown at WebSphere Administration Command view.

With the Jython script open in the Jython editor, and the monitored command data in the WebSphere Administration Command view, you can insert the commands directly into the script file. Place the cursor in the script file where you want the command to go. Then right-click the command and select **Insert**, as shown in Figure 5 on page 38. Double-clicking the command also inserts it into the script.



```
#
# TODO: enter JYTHON code and saveAdminControl.invoke(\
   'WebSphere:name=test,process=dmgr,platform=common,noc
   'start')

#cleate cluster
AdminTask.createCluster(\
   '[-clusterConfig [-clusterName test -preferLocal true
AdminTask.createClusterMember(\
   '[-clusterName test -memberConfig [-memberNode sys1Nc
AdminTask.createClusterMember(\
```

Description	Command	Time
WASX7056I: Method: list	Admin	2/24/09 2:03 ...
Creates a new application ser...	Admin	2/24/09 2:04 ...
Creates a new member of an ...	Admin	2/24/09 2:04 ...
Creates a new member of an ...	Admin	2/24/09 2:04 ...
WASX7056I: Method: list	Admin	2/24/09 2:04 ...
WASX7124I: Method: save	AdminConfig.save()	2/24/09 2:04 ...

Figure 5 Jyton editor running on automatically the command output

Example: Using scripts with the job manager

This section provides an example of how to use scripting to automate a WebSphere installation that uses a flexible management environment.

Most companies have routine tasks that occur in different phases of the software development life cycle. In an environment with multiple WebSphere Application Server installations, automation of these tasks can save a lot of time. The combination of wsadmin scripts, script libraries, and the automated management provided in a flexible management environment provides an automation solution.

Introduction

The scenario uses a simple WebSphere environment to illustrate how to automate tasks. The techniques here can be used in much more complex environments. This scenario contains three steps:

1. Write a customized script to automate the tasks
2. Configure the job manager
3. Verify the results

Figure 6 shows the scenario environment. A single application server is configured server on Node B. The deployment manager for the cell is registered to a job manager on Node A.

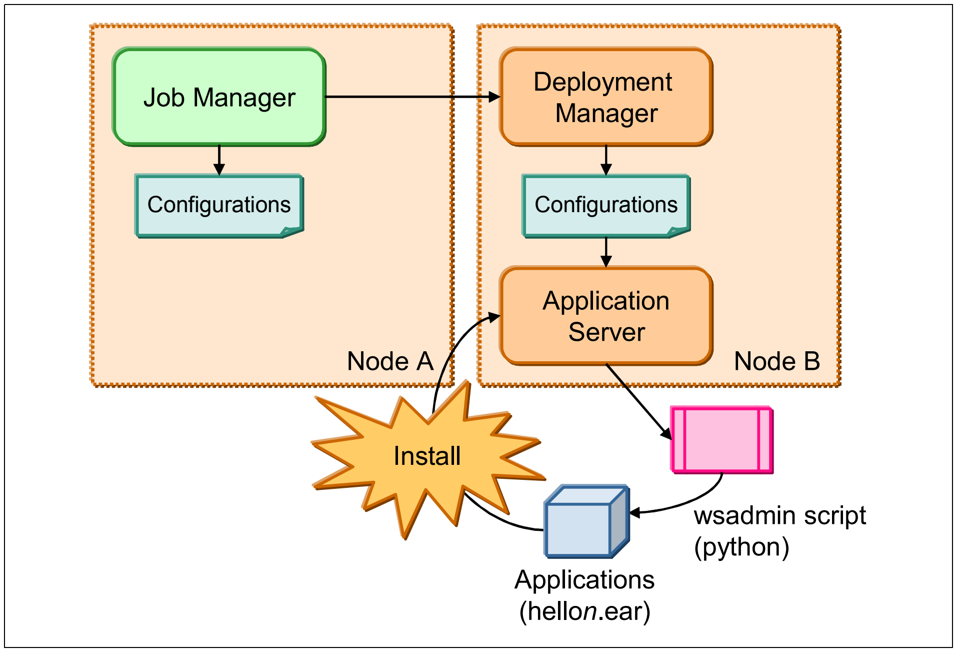


Figure 6 the environment details

Applications in this environment are installed frequently and the administrator needs a quick way to install these applications. Here is the plan:

1. A wsadmin script is prepared to install an application. The script will make use of the script libraries.
2. A job is scheduled to run the script at regular intervals.

3. The script checks a text file that lists the new applications to be installed.

   The new application information stored in a text file includes the application name, application location, node name, and server name (see Figure 7).

4. If the job runs and finds that the application is not installed, the application will be installed. If the application is already installed, it will be uninstalled, then re-installed.

5. The text file is renamed "*filename*.txt.old" whenever the job is executed.

6. If the job executes and no text file exists, no actions are taken. It is only when you distribute a new text file and application that the job will perform the install.

To test, two applications are prepared; hello1.ear and hello2.ear.

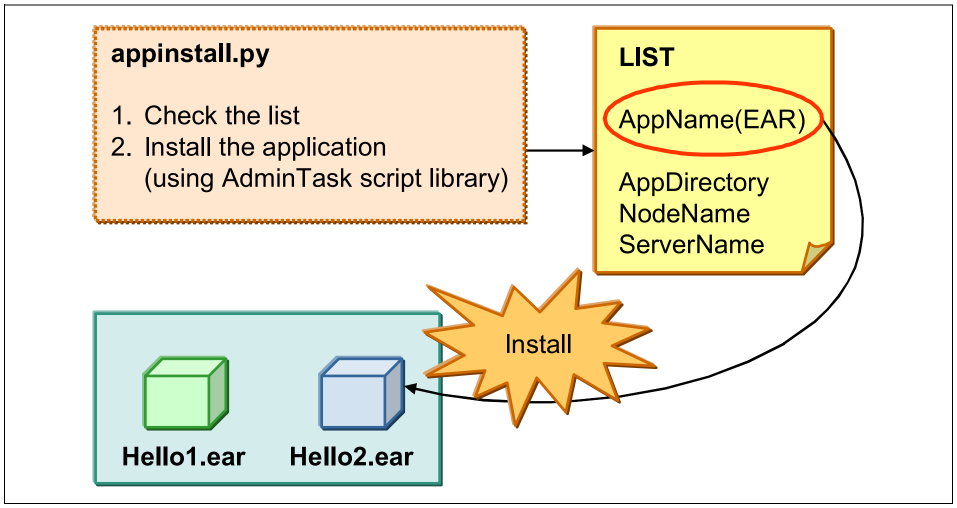


Figure 7 Scripting details

Creating the customized script

The AdminApplication script in the script libraries has been identified as having procedures that will accomplish the installation and update of applications.

In this scenario, two procedures from the script libraries were identified for use:

- `AdminApplication.uninstallApplication`

  This procedure will be used to uninstall an application. The arguments specify the application name.

- AdminApplication.installAppWithNodeAndServerOptions:

  This procedure will be used to install an application. This procedure is selected because there is only a single server in this environment. In an environment with clustered application servers, the installAppWithClusterOption procedure can be used instead.

The script file is written in Python and is called appInstall.py (see Example 28).

Example 28 appInstall.py

```
#Check the list and intall/update the applications.
#

import sys
import java
import os
import re
import time
from java.lang import *

dir = "C:/WebSphereV7/AppServer/profiles/dmgr40/downloadedContent/inputfile"
#
sep = System.getProperty("line.separator")
for fname in os.listdir(dir):
   print fname
   path = os.path.join(dir, fname)
   if (os.path.isfile(path)) and (not re.match(".*old$",path) and (not 
re.match("appInstall.py",fname))):
      print "procesing %s" % (path)
      inp = open(path, 'r')
      for line in inp.readlines():

         itemList = line[:-1].split(',')
         appName = itemList[0]
         earFile = itemList[1]
         nodeName= itemList[2]
         serverName = itemList[3]

         # application existence check
         print "application existence check"
         existAppList = AdminApp.list().split(sep)
         isExist = 0
         for existApp in existAppList:
            if(appName == existApp):
               isExist = 1
               break
         # acquire application manager
         print "acquire application manager"

            appMgrID =
AdminControl.queryNames("type=ApplicationManager,node="+nodeName+",process="+serverNa
me+",*" )

         # if exist, uninstall application
         print "app exists - uninstall"
         if( isExist == 1 ):
            appId = ""
            try:
                _excp_ = 0
                appID =
AdminControl.completeObjectName("type=Application,node="+nodeName+",Server="+serverNa
me+",name="+appName+",*" )
               except:
                _type_, _value_, _tbck_ = sys.exc_info()
                _excp_ = 1
               #endTry
            # if running, stop application
            print "appID is %s" % (appID)
            if(len(appID) > 0):
               print "stopping %s" % (appName)
               stopped  = AdminControl.invoke(appMgrID, "stopApplication", appName)
               sleep(1)
            # uninstall application
            print "Uninstalling %s" % (appName)
            AdminApplication.uninstallApplication(appName)

         # install application
         print "Installing %s" % (appName)
         AdminApplication.installAppWithNodeAndServerOptions(appName, earFile,
nodeName, serverName)
         print "Starting %s" % (appName)
         started  = AdminControl.invoke(appMgrID, "startApplication", appName)
         sleep(1)

      inp.close()
      os.rename(fname, fname + ".old")
   #endIf
#endFor
```

The input file, hello.txt, is shown in Example 29.

Example 29 hello.txt

```
hello,/webspherev7/appserver/profiles/dmgr40/downloadedContent/appl/hello1.ear,node40
a,server40a1
```

The sample script is first tested using wsadmin running in script mode. The result of the sample script is shown in Example 30.

Example 30 testing the sample script

```
C:\WebSphereV7\AppServer\profiles\dmgr40\bin>wsadmin -lang jython -f c:\webspher
ev7\appserver\profiles\dmgr40\downloadedContent\appinstall.py -username admin -p
assword admin
WASX7209I: Connected to process "dmgr" on node dmgr40node using SOAP connector;
 The type of process is: DeploymentManager
hello.txt
procesing C:\WebSphereV7\AppServer\profiles\dmgr40\downloadedContent\inputfile\h
ello.txt
application existence check
acquire application manager
app exists - uninstall
Installing hello
---------------------------------------------------------------
 AdminApplication:       Install application with -node and -server options
 Application name:       hello
 Ear file to deploy:     /webspherev7/appserver/profiles/dmgr40/downloadedConten
t/appl/hello1.ear
 Node name:              node40a
 Server name:            server40a1
 Usage: AdminApplication.installAppWithNodeAndServerOptions("hello", "/websphere
v7/appserver/profiles/dmgr40/downloadedContent/appl/hello1.ear", "node40a", "ser
ver40a1")
---------------------------------------------------------------


ADMA5016I: Installation of hello started.
ADMA5058I: Application and module versions are validated with versions of deploy
ment targets.
ADMA5005I: The application hello is configured in the WebSphere Application Serv
er repository.
ADMA5053I: The library references for the installed optional package are created
.
ADMA5005I: The application hello is configured in the WebSphere Application Serv
er repository.
```

```
ADMA5001I: The application binaries are saved in C:\WebSphereV7\AppServer\profil
es\dmgr40\wstemp\Script120f8e64a06\workspace\cells\Cell40\applications\hello.ear
\hello.ear
ADMA5005I: The application hello is configured in the WebSphere Application Serv
er repository.
SECJ0400I: Successfully updated the application hello with the appContextIDForSe
curity information.
ADMA5005I: The application hello is configured in the WebSphere Application Serv
er repository.
CWSAD0040I: The application hello is configured in the Application Server reposi
tory.
ADMA5113I: Activation plan created successfully.
ADMA5011I: The cleanup of the temp directory for application hello is complete.
ADMA5013I: Application hello installed successfully.
OK: installAppWithNodeAndServerOptions('hello', '/webspherev7/appserver/profiles
/dmgr40/downloadedContent/appl/hello1.ear', 'node40a', 'server40a1', 'false'):
```

Submitting the job

This section describes how to use the job manager to execute the script:

1. Before running the appInstall.py script, you must transfer it, along with the hello.txt file and the hello1.ear file from the job manager to the deployment manager using the distributeFile job (see Chapter 5, *Administration consoles and commands* to define the directories required for this task).

   In Rational Application Developer, export the appInstall.py script file and application EAR file to the *jmgr_profile_root*/config/temp/JobManager directory.

   Manually copy the hello.txt file to the same directory.

2. In the Job manager console, select the **Job** → **Submit** menu. This will launch the Job properties wizard.

   Use the **Distribute file** job to transfer each file. In each case, select the admin agent as the target node.

   The source location refers to the file in the *jmgr_profile_root*/config/temp/JobManager directory.

   The format is:

   `file:/`*`file_name`*

   The hello.txt file should be distributed to the following directory:

   `dmgr_profile_root``/downloadedContent/inputfile`

The appInstall.py file should be distributed to the following directory:

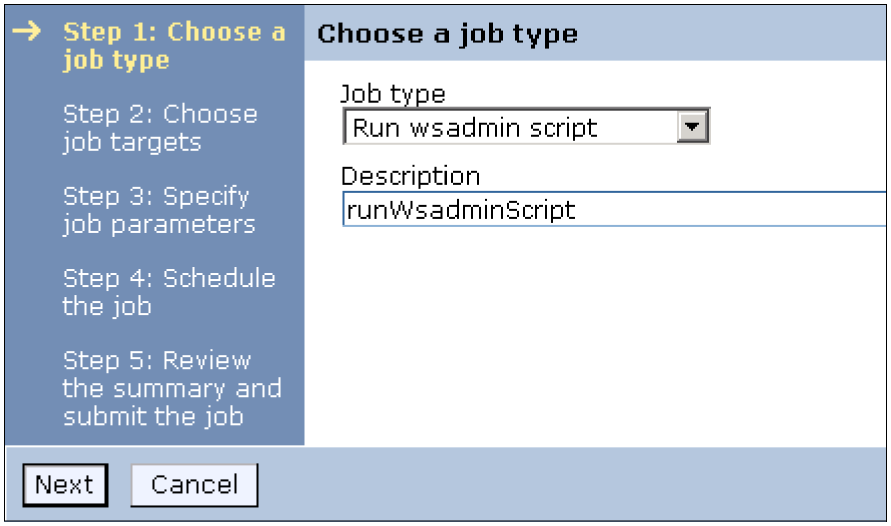
dmgr_profile_root`/downloadedContent`

The appInstall.py file should be distributed to the following directory:

dmgr_profile_root`/downloadedContent/appl`

3. The next step is to submit the wsadmin script for execution. Select the **Job** → **Submit** menu. This will launch the Job properties wizard.

   a. Select **Run wsadmin script**. as the job type and enter a description. Click **Next** (Figure 8).


→ Step 1: Choose a job type
Step 2: Choose job targets
Step 3: Specify job parameters
Step 4: Schedule the job
Step 5: Review the summary and submit the job
Next
Cancel

Choose a job type

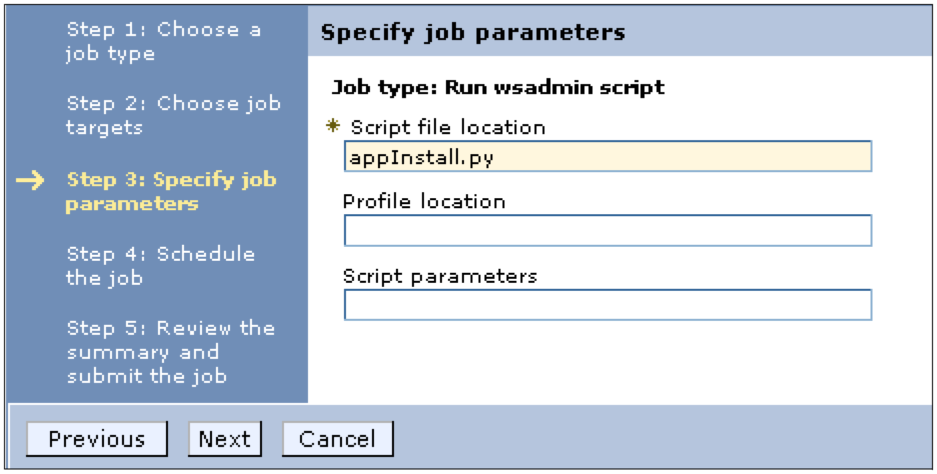
Job type: Run wsadmin script

Description: runWsadminScript

Figure 8   Step1: Choose the job type

   b. Select the job target. In this case, the deployment manager node is selected. Enter the user ID and password required for administration tasks on the deployment manager. Click **Next.**

   c. Specify the script location. This is the same location you used when you distributed the file. The current directory is *dmgr_profile_root/*downloadedContent directory. Then, click **Next** (Figure 9).


Step 1: Choose a job type
Step 2: Choose job targets
Step 3: Specify job parameters
Step 4: Schedule the job
Step 5: Review the summary and submit the job
Previous
Next
Cancel

Specify job parameters

Job type: Run wsadmin script

* Script file location: appInstall.py

Profile location:

Script parameters:

Figure 9 Step3: Specify job parameters

   d. The next step allows you to schedule the job to run once, or at regular intervals. It also allows you to define when the job is available for execution and when it expires.

      In this example, the execution of the script from the job manager will be tested first using the Run once option in the Availability interval field. After the job has been tested, the job submit process can be repeated and an interval can be set so the job runs automatically.

      Click **Next**.

   e. Review the summary and click **Finish**.

4. Monitor the job status to ensure it completes successfully.
5. If the job does not complete successfully, click the Job ID to see the messages produced.

Check the results

The results can be verified by displaying the new application in the administrative console, starting it, then accessing it from a Web browser:

1. Access the deployment manager console and expand **Applications** from the navigation tree and click **Application Types** → **WebSphere enterprise** application.
2. Verify that the new application is in the list.

Online resources

These Web sites and URLs are also relevant as further information sources:

- Command assistance simplifies administrative scripting in WebSphere Application Server:

  http://www.ibm.com/developerworks/websphere/library/techarticles/0812_rhodes/0812_rhodes.html

- Sample Scripts for WebSphere Application Server Versions 5 and 6:

  http://www.ibm.com/developerworks/websphere/library/samples/SampleScripts.html

Notices

This document REDP-4576-00 was created or updated on October 12, 2009.

Send us your comments in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099, 2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Redpaper™

Trademarks