



*WebSphere Application  
Server V8.5 Scripting and  
Automation*

(Course code WA680 / VA680)

**Student Exercises**

ERC 1.0

Authorized

**IBM | Training**

WebSphere Education

## Trademarks

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AFS™	AIX®	DB™
DB2®	Express®	HACMP™
IMS™	MVS™	RACF®
Rational®	RDN®	Tivoli®
WebSphere®	WPM®	z/OS®
zSeries®		

Intel and Intel Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

### May 2013 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

# Contents

<b>Trademarks .....</b>	<b>v</b>
<b>Exercises description .....</b>	<b>vii</b>
<b>Verifying the image and course materials .....</b>	<b>ix</b>
<b>Exercise 1. Using wsadmin to explore the WebSphere Application Server environment .....</b>	<b>1-1</b>
<b>Exercise 2. Using the IBM Assembly and Deploy Tools (IADT) to develop Jython scripts .....</b>	<b>2-1</b>
<b>Exercise 3. Using the Help and AdminConfig objects .....</b>	<b>3-1</b>
<b>Exercise 4. Using the AdminApp object .....</b>	<b>4-1</b>
<b>Exercise 5. Using the AdminControl object .....</b>	<b>5-1</b>
<b>Exercise 6. Using the AdminTask object .....</b>	<b>6-1</b>
<b>Exercise 7. Creating and configuring the Plants server environment with scripting .....</b>	<b>7-1</b>
<b>Exercise 8. Installing and configuring the IBM HTTP Server .....</b>	<b>8-1</b>
<b>Exercise 9. Deploying the PlantsByWebSphere application .....</b>	<b>9-1</b>
<b>Exercise 10. ws_ant scripting and configuring the service integration bus .....</b>	<b>10-1</b>
<b>Exercise 11. Automating the installation of WebSphere Application Server .....</b>	<b>11-1</b>



# Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AFS™	AIX®	DB™
DB2®	Express®	HACMP™
IMS™	MVS™	RACF®
Rational®	RDN®	Tivoli®
WebSphere®	WPM®	z/OS®
zSeries®		

Intel and Intel Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.



# Exercises description

This course includes the following exercises:

- Exercise 1. Using wsadmin to explore the WebSphere Application Server environment
- Exercise 2. Using the IADT to develop Jython scripts
- Exercise 3. Using the AdminConfig object
- Exercise 4. Using the AdminApp object
- Exercise 5. Using the AdminControl object
- Exercise 6. Using the AdminTask object
- Exercise 7. Creating and configuring the Plants server environment by using scripting
- Exercise 8. Creating and configuring the IBM HTTP Server and web server plug-in
- Exercise 9. Deploying and managing the PlantsByWebSphere application by using scripting
- Exercise 10. ws\_ant scripting and configuring the service integration bus
- Exercise 11. Automating the installation of WebSphere Application Server

In the exercise instructions, each step has a check offline. You might want to check off each step as you complete it to track your progress.

Most exercises include required sections that should always be completed. These sections might be required before performing later exercises. Some exercises might also include optional sections that you might want to perform if you have sufficient time and want an extra challenge.

Make sure that you know the machine's host name because you need it in the exercises. The name has a format of 'was85host'.

The following Linux administrator user ID is created for you and must be used to log on to the system:

- User ID: **root**
- Password: **web1sphere**

The Linux passwords are case-sensitive.

Variables are used throughout the exercises to indicate part of a path name. When you encounter a <variable> tag, use the following table to determine the correct value for the variable.

<variable>	Value
<db2_root>	/opt/ibm/db2/V9.7
<hostname>	was85host
<ihs_root>	/opt/IBM/HTTPServer
<lab_work>	/usr/LabWork
<nodename>	was85hostNode01
<password>	web1sphere
<plugin_root>	/opt/IBM/WebSphere/Plugins
<profile_root>	/opt/IBM/WebSphere/AppServer/profiles
<software_dir>	/usr/Software
<solutions_dir>	/usr/Solutions
<userid>	root
<was_root>	/opt/IBM/WebSphere/AppServer

-

# Verifying the image and course materials

The student books and VMware image for this course display a release number that is called the edition revision code (ERC). To verify that the books and image are at the same level, compare the ERC of the VMware image to the ERC of the student books.

- \_\_\_ 1. Determine the ERC number of your course materials.
  - \_\_\_ a. Open all of the books you received (either printed or PDF).
  - \_\_\_ b. Note the ERC listed on the front page of your books. The ERC number is listed under the course title on the first page of the books as in the following example:

*Administration of IBM  
WebSphere Process Server  
V7*

(Course code WB722 / VB722)

## Student Exercises

ERC 1.0

- \_\_\_ 2. Determine the ERC number of the VMware image.
  - \_\_\_ a. On the image desktop, open the **readme.txt** file.
  - \_\_\_ b. Note the ERC listed in the file. The ERC number is indicated on the “ERC number” line as in the following example:

```
-----
-----
Student workstation virtual Machine Information
Websphere Education

Copyright (C) 2011 IBM Corporation
Course materials, including this virtual machine, may not be
reproduced in
whole or in part without prior written permission of IBM.
-----

Course code: WB722/VB722
Course title: Administration of IBM websphere
Process Server V7
Application Server
ERC number: 1.0
Last modified date: April 26, 2011

Operating system: Microsoft windows XP Professional SP3
Primary account username: Administrator
Primary account password: web1sphere
```

- \_\_\_ 3. Verify the image and course materials.
  - \_\_\_ a. If the ERC number on the image does not match the ERC number on the printed materials, notify your instructor that the materials are not synchronized.



**Stop**

Do NOT proceed with the exercises if the ERC numbers on the course materials and course image do not match; ask your instructor for further direction.



# Exercise 1. Using wsadmin to explore the WebSphere Application Server environment

## What this exercise is about

In this exercise, you learn how to start and use the wsadmin tool to explore the installed WebSphere Application Server environment. All methods to start wsadmin are used. The help features are explored, and the administrative objects are used along with simple Jython operations to query configuration data.

## What you should be able to do

At the end of this exercise, you should be able to:

- Start wsadmin in its three supported modes
- Run simple commands that use the Jython syntax
- Run simple commands that use the Administrative objects to query the WebSphere Application Server environment

## Introduction

WebSphere Application Server provides the wsadmin tool to manage and configure resources, without using a graphical user interface, through scripting. The wsadmin tool allows administrators to reuse completed scripts, which ensure consistency and reduce the chance of human error. These scripts can also be used to quickly rebuild an environment if there is a critical error, or to build a duplicate of an environment with simple changes to property files.

The Jython language for the wsadmin tool is the strategic direction for WebSphere Application Server administrative automation. Beginning in the V6.1 release, WebSphere Application Server contains significantly enhanced administrative functions and tools that support product automation and the use of the Jython language.

In the first part of this exercise, you review the startup options and help facilities available for wsadmin. You update the *wsadmin.properties* file to set the default language to Jython. You also start the wsadmin tool to run single scripts and script files. Finally, you use the *execfile()* method to quickly run scripts from wsadmin in interactive mode.

## Requirements

To complete this exercise, you must have the WebSphere Application Server Network Deployment V8.5 product installed and a server named *server1* created in the *SamplesProfile* profile. The Default application must also be deployed on *server1*.



### Information



## Exercise instructions



### Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

`<was_root>`: /opt/IBM/WebSphere/AppServer

`<profile_root>`: /opt/IBM/WebSphere/AppServer/profiles

### Section 1: Starting wsadmin

The wsadmin tool can be used to manage the configuration and runtime operation of WebSphere Application Server. In this section, you explore the startup options for wsadmin, review and modify the `wsadmin.properties` file, and then run simple commands by using the Jython language to familiarize yourself with wsadmin. In subsequent steps, you use the interactive and script execution functions.

- 1. Start WebSphere Application Server.
  - a. Open a terminal window by clicking the **GNOME Terminal** icon.
  - b. Go to the `<profile_root>/SamplesProfile/bin` directory.
  - c. Enter the following command to start the server1:

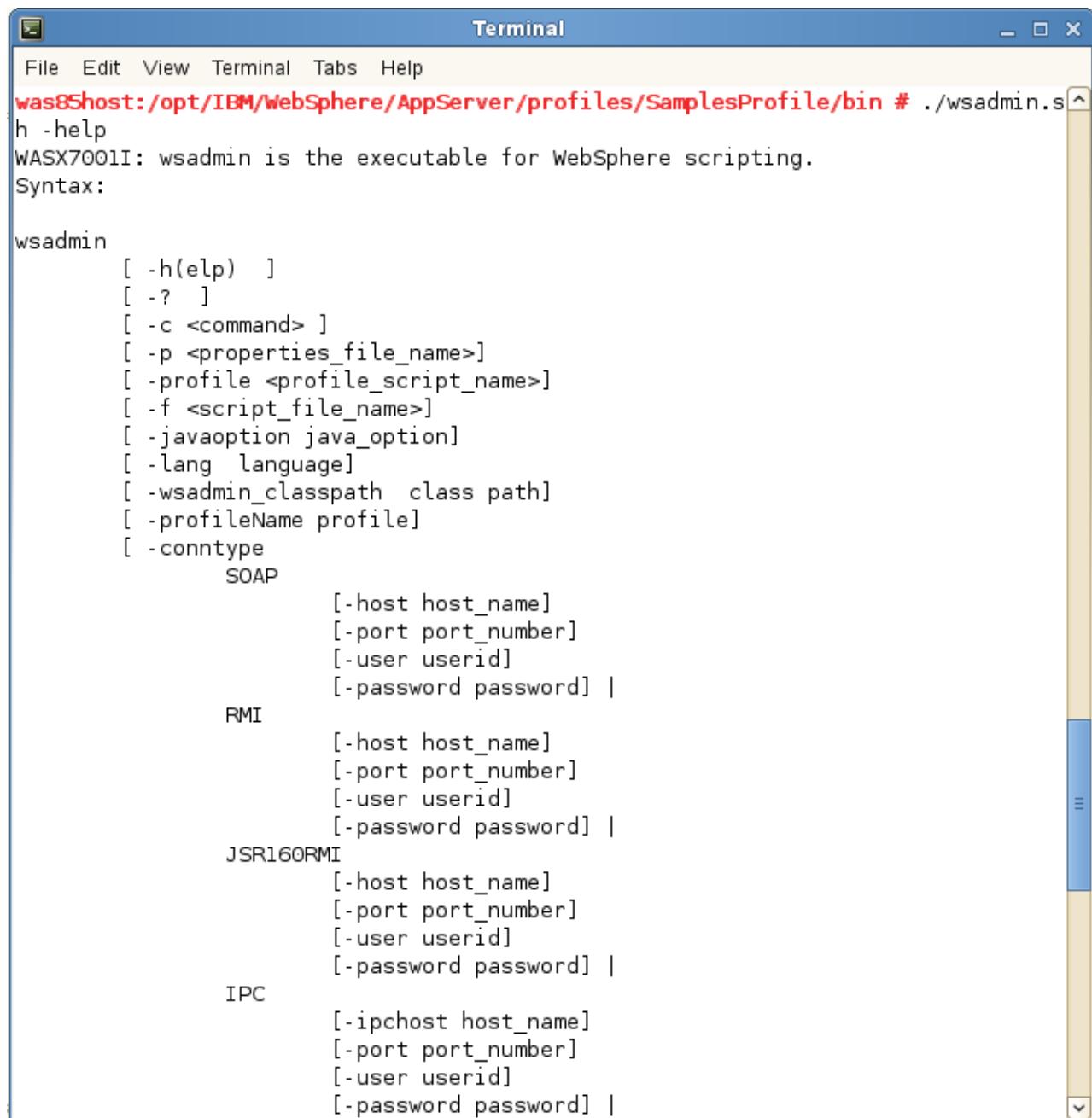
`./startServer.sh server1`

```

Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./startServer.sh server1
ADMU0116I: Tool information is being logged in file
          /opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/logs/server1/startServer.log
ADMU0128I: Starting tool with the SamplesProfile profile
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server server1 open for e-business; process id is 13667
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin #
  
```

- \_\_\_ 2. List the command-line options for wsadmin.
- \_\_\_ a. In the terminal window, enter the following command to get the command-line options for wsadmin:

```
./wsadmin.sh -help
```



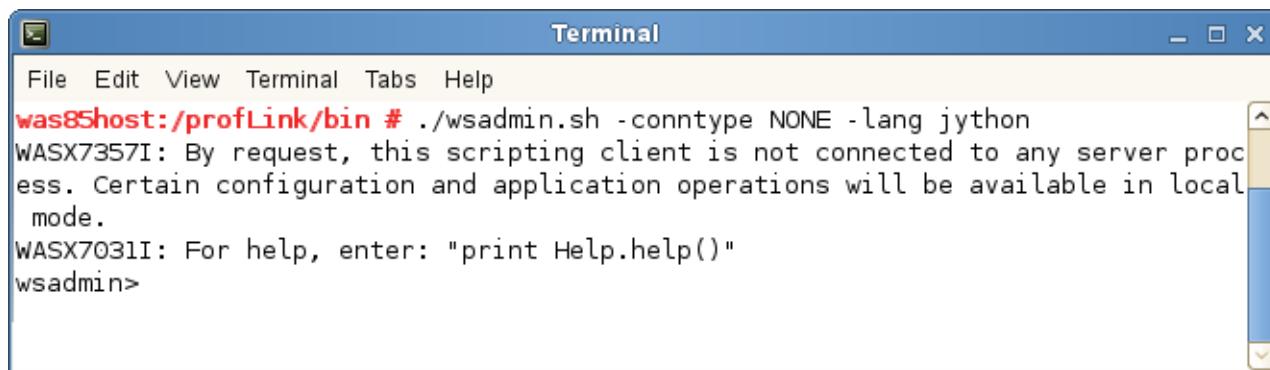
The screenshot shows a terminal window titled "Terminal". The command `./wsadmin.sh -help` was run, and the output is displayed. The output includes a brief description of wsadmin as the executable for WebSphere scripting, the syntax for running wsadmin, and detailed options for different communication protocols: SOAP, RMI, JSR160RMI, and IPC.

```
File Edit View Terminal Tabs Help  
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./wsadmin.sh -help  
WASX7001I: wsadmin is the executable for WebSphere scripting.  
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 ] |
```

The resulting output displays a list of command-line options and the full description of their usage.

- \_\_\_ 3. Connect to wsadmin by using the **NONE** connection type. The **-conntype NONE** option runs wsadmin in local mode. The result is that the scripting client is not connected to a running server. However, you can still manage server configuration, and the installation and the uninstallation of applications when the application server is not running. Begin by starting wsadmin.
- \_\_\_ a. In the command prompt, run the following command to start wsadmin in interactive mode:

```
./wsadmin.sh -conntype NONE -lang jython
```



A screenshot of a Windows terminal window titled "Terminal". The window shows the command `./wsadmin.sh -conntype NONE -lang jython` being run. The output message is: "WASX7357I: By request, this scripting client is not connected to any server process. Certain configuration and application operations will be available in local mode." Another message follows: "WASX7031I: For help, enter: "print Help.help()"". The prompt "wsadmin>" is visible at the bottom.



### Note

When using wsadmin with a connection type of **NONE**, the *username* and *password* parameters are not required.



### Note

When wsadmin is started with **-lang jython** for the first time in a profile, the Jython language libraries get loaded into the environment. A long series of informational messages are produced on the wsadmin console.

- \_\_\_ 4. Get familiar with the Help administrative object. The connection type of **NONE** allows you to run any of the help commands. Use the Help administrative object and the `help()` method to help you understand the methods and capabilities of the administrative objects.



## Information

The Help administrative object provides online information about running MBeans, and help messages. Use the Help administrative object to obtain general help for the AdminApp, AdminConfig, AdminTask, and AdminControl administrative objects. The Help administrative object also provides interface information about MBeans running in the system.

- \_\_ a. Enter the following command to list the available commands for the Help object:

```
print Help.help()
```

The screenshot shows a terminal window titled "Terminal". The window contains the following text:

```
File Edit View Terminal Tabs Help
wsadmin>print Help.help()
WASX7028I: The Help object has two purposes:
First, provide general help information for the the objects
supplied by wsadmin for scripting: Help, AdminApp, AdminConfig,
AdminControl and AdminTask.
Second, provide a means to obtain interface information about
MBeans running in the system. For this purpose, a variety of
commands are available to get information about the operations,
attributes, and other interface information about particular
MBeans.

attributes           Given an MBean, returns help for attributes
operations          Given an MBean, returns help for operations
constructors        Given an MBean, returns help for constructors
description         Given an MBean, returns help for description
notifications       Given an MBean, returns help for notifications
classname           Given an MBean, returns help for classname
all                 Given an MBean, returns help for all the above
help                Returns this help text
AdminControl         Returns general help text for the AdminControl object
AdminConfig          Returns general help text for the AdminConfig object
AdminApp             Returns general help text for the AdminApp object
AdminTask            Returns general help text for the AdminTask object
wsadmin              Returns general help text for the wsadmin script
launcher
message             Given a message id, returns explanation and user
action message
```

The list of the available help methods are displayed. You can retrieve information such as attributes, operations, and descriptions when developing your Jython scripts.

- \_\_\_ b. At the wsadmin command prompt, enter the following commands for an overview of the capabilities for the administrative objects and a list of available methods:

```
print Help.AdminApp()  
print Help.AdminControl()  
print Help.AdminConfig()  
print Help.AdminTask()
```

- \_\_\_ c. More detailed information about each command is available by using the help() method. Enter the following command for more information about the edit() method of the AdminApp object:

```
print AdminApp.help('edit')
```

A screenshot of a Windows terminal window titled "Terminal". The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The command "wsadmin>print AdminApp.help('edit')" is entered, followed by the output: "WASX7104I: Method: edit". Below this, the arguments and description for the edit method are displayed.

```
File Edit View Terminal Tabs Help  
wsadmin>print AdminApp.help('edit')  
WASX7104I: Method: edit  
  
Arguments: application name, options  
  
Description: Modifies the application specified by "application name" using the options specified by "options". The user is not prompted for any information.  
wsadmin>
```

- \_\_\_ d. Use the message() method from the Help object to obtain information and possible solutions to warning and error messages. For example, you might receive a **WASX7115E** error when running the AdminApp.install() method.

In the terminal window, enter the following command:

```
print Help.message('WASX7115E')
```

A screenshot of a Windows terminal window titled "Terminal". The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The command "wsadmin>print Help.message('WASX7115E')" is entered, followed by the output: "Explanation: The wsadmin tool cannot read the ear file when preparing to copy it to a temporary location for AdminApp processing." and "User action: Examine the wsadmin.traceout log file to determine the problem. Verify that the file path is correct and that there are no file permission problems.". Below this, the wsadmin prompt is shown.

```
File Edit View Terminal Tabs Help  
  
wsadmin>print Help.message('WASX7115E')  
Explanation: The wsadmin tool cannot read the ear file when preparing to copy it to a temporary location for AdminApp processing.  
User action: Examine the wsadmin.traceout log file to determine the problem. Verify that the file path is correct and that there are no file permission problems.  
wsadmin>
```

The message that is returned provides an explanation for the **WASX7115E** error. In this case, the EAR file that is specified in the installation method can not be found.

- \_\_\_ 5. The wsadmin connection type *NONE* can also be used to manage installed applications. List the installed applications and export the DefaultApplication application.
- \_\_\_ a. Run the following command to list the installed applications:

```
print AdminApp.list()
```



The terminal window shows the command `print AdminApp.list()` being run. The output lists several installed applications: DefaultApplication, ivtApp, query, and wsadmin. The window has a standard Windows-style title bar and scroll bars.

```
File Edit View Terminal Tabs Help
wsadmin>print AdminApp.list()
DefaultApplication
ivtApp
query
wsadmin>
```

The list of installed applications is returned.

- \_\_\_ b. Run the following command to export the DefaultApplication:

```
AdminApp.export('DefaultApplication','/tmp/DefaultApplication.ear')
```



The terminal window shows the command `AdminApp.export('DefaultApplication','/tmp/DefaultApplication.ear')` being run. The output shows two blank lines, indicating a successful execution. The window has a standard Windows-style title bar and scroll bars.

```
File Edit View Terminal Tabs Help
wsadmin>AdminApp.export('DefaultApplication','/tmp/DefaultApplication.ear')
''
wsadmin>
```

The DefaultApplication is packaged and exported as an EAR file to the specified directory. Although no descriptive message was returned, the command completed successfully. If the command fails to run, an error message is displayed.

- \_\_\_ c. Enter the following command to exit the wsadmin interactive mode:

```
exit or quit
```



The terminal window shows the command `exit` being run. The output shows the path `was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin #`, indicating the user has exited the wsadmin session. The window has a standard Windows-style title bar and scroll bars.

```
File Edit View Terminal Tabs Help
wsadmin>
wsadmin>exit
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin #
```

## Section 2: Configuring wsadmin properties and profile scripts

The wsadmin tool, upon startup, reads a set of property files to manage and customize the scripting environment.

The property files include specifications for the connection type, the port, and host that is used when attempting a connection. Also included in property files are the location where trace and logging output are directed, and the temporary directory to access files while installing applications and modules. Also, the class path information to append to the list of paths to search for classes and resources is included in property files.

You can make temporary alterations to the scripting environment with the **-profile** option. This option is used to run one or more script files when you start the scripting tool. In addition, the **-p** is used to specify scripting properties that are defined in a file.

The wsadmin scripting utility loads the following levels of property files:

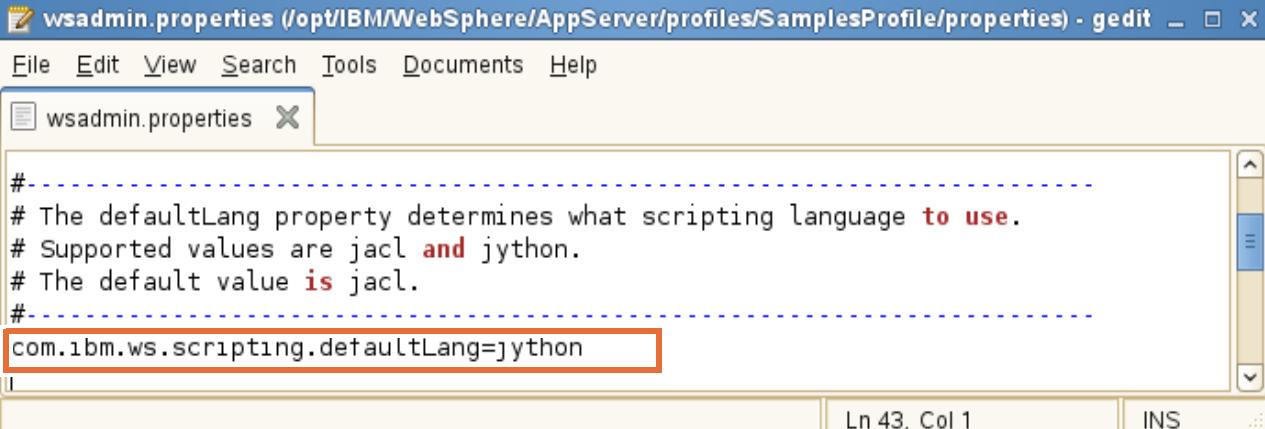
- The files that are specified by the WSADMIN\_PROPERTIES environment variable.
- The <was\_root>/properties/wsadmin.properties file.
- The <profile\_root>/<profile\_name>/properties/wsadmin.properties file.
- Any property files that are specified on the command line.



### Important

The properties files load in the order that is listed. The property file that is loaded last takes precedence over the ones that are loaded before it.

- 1. Edit the `wsadmin.properties` file and update the wsadmin default language from Jacl to Jython.
  - a. Use a file system browser such as Nautilus to navigate to the `<profile_root>/SamplesProfile/properties` directory.
  - b. Open the `wsadmin.properties` file by using a text editor.
  - c. Although Jython is now the preferred scripting language, the default language that is specified is still set to `jacl`. Locate the property `com.ibm.ws.scripting.defaultLang`.
  - d. Set the `com.ibm.ws.scripting.defaultLang` to `jython`.



```
#  
# The defaultLang property determines what scripting language to use.  
# Supported values are jacl and jython.  
# The default value is jacl.  
  
#  
com.ibm.ws.scripting.defaultLang=jython
```

Ln 43, Col 1      INS

- \_\_\_ e. Save the changes (**Ctrl+S**) and close the file.



#### Note

After the default language property is set to **jython**, the **-lang jython** startup option is no longer required.

- \_\_\_ 2. Attempt to start wsadmin by connecting to a server, and observe that you are prompted to enter a user ID and password.

- \_\_\_ a. Enter: **./wsadmin.sh**

You are prompted because the server was installed, and are running with administrative security enabled; therefore you must authenticate before a connection can be established.

- \_\_\_ b. In the authentication dialog, enter **wasadmin** for the User Identity and **websphere** for the User Password. Click **OK**.
  - \_\_\_ c. After wsadmin starts, type **exit** to quit.

You can supply the user ID and the password on the command line by using the format:

```
./wsadmin.sh -username wasadmin -password websphere
```

This format is generally considered insecure since the user and password information are in clear text and are available to be recalled through command history.

Entering the user and password in the dialog, as you did in this example can also be inconvenient especially when running in unattended mode and a human is not available to enter this information.

In the next step, you store this information in a properties file, and therefore avoid having to enter it manually.

- \_\_\_ 3. Hardcode the user ID and password to avoid entering them on the wsadmin command line, or being prompted in a window. Update the `soap.client.props` file with the correct user name and password values.



### Warning

In a production environment, setting these values in the property files, even if encoded, does not provide sufficient security. File system protection, at the operating system level, must also be configured to ensure appropriate protection of these important files. To encode the properties, refer to **Section 6**.

- \_\_\_ a. Using the file system browser, go to the `<profile_root>/SamplesProfile/properties` directory.
- \_\_\_ b. Open the `soap.client.props` file by using a text editor.
- \_\_\_ c. Locate the `com.ibm.SOAP.loginUserId` property and set the value to `wasadmin`.
- \_\_\_ d. Locate the `com.ibm.SOAP.loginPassword` property and set the value to `websphere`.

```
*soap.client.props (/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/properties) _ □ X
File Edit View Search Tools Documents Help
*soap.client.props X

#-----
# SOAP Client Security Enablement
# - security enabled status ( false[default], true )
#-----
com.ibm.SOAP.securityEnabled=false

#-----
# - authenticationTarget ( BasicAuth[default], KRB5.
# These are the only supported selection
# on a pure client for JMX SOAP Connector Client. )
#-----
com.ibm.SOAP.authenticationTarget=BasicAuth

com.ibm.SOAP.loginUserId=wasadmin
com.ibm.SOAP.loginPassword=websphere
```

The user name and password parameters are no longer required when invoking wsadmin with the SOAP connection type.

- \_\_ e. Save the changes (**Ctrl+S**) and close the file.



### Note

The default connection type for wsadmin is SOAP. If you must connect by using RMI and you want to hardcode the user name and password values, you must update the **sas.client.props** file, which is in the **<profile\_root>/SamplesProfile/properties** directory.

- \_\_ f. Verify you can connect to the server1 process by entering the following command:

```
./wsadmin.sh
```

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./wsadmin.sh
WASX7209I: Connected to process "server1" on node was85hostNode01 using SOAP connector: The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>
```

You successfully connected to wsadmin and no longer must provide the user name and password parameters on the command line.

- \_\_ g. Type **exit** to exit wsadmin.
- \_\_ 4. When wsadmin is run, profile scripts are executed before any other scripts. A profile script is a script that runs before the main script or before you enter interactive mode.

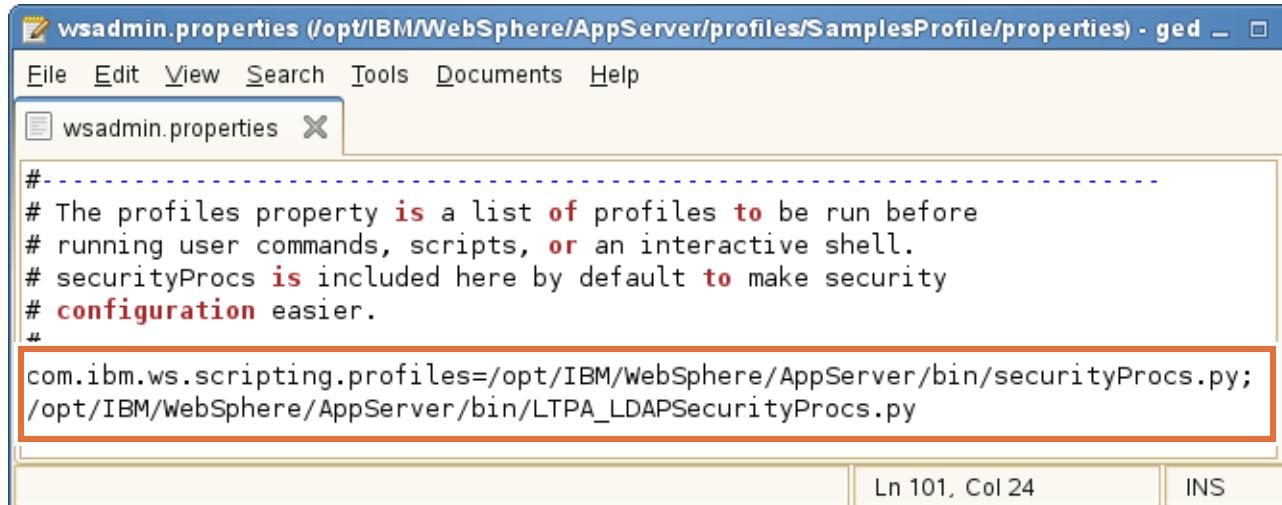


### Important

The **com.ibm.ws.scriptingprofiles** property in the **wsadmin.properties** file sets the profile scripts that load on startup. By default, the profile scripts are set to **securityProcs.jacl** and **LTPA\_LDAPSecurityProcs.jacl**. These files are Jacl scripts and must be changed to reflect the new Jython language setting.

Change the `com.ibm.ws.scriptingprofiles` property to reflect the default language change from Jacl to Jython.

- \_\_ a. Using a file system browser, go to the `<profile_root>/SamplesProfile/properties` directory.
- \_\_ b. Open the `wsadmin.properties` file by using a text editor.
- \_\_ c. Locate the `com.ibm.ws.scriptingprofiles` property.
- \_\_ d. You must change only the file types from `jac1` to `py` as follows:  
`/opt/IBM/WebSphere/AppServer/bin/securityProcs.py;`  
`/opt/IBM/WebSphere/AppServer/bin/LTPA_LDAPSecurityProcs.py`



```
wsadmin.properties (/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/properties) - ged ...
File Edit View Search Tools Documents Help
wsadmin.properties X
#
# The profiles property is a list of profiles to be run before
# running user commands, scripts, or an interactive shell.
# securityProcs is included here by default to make security
# configuration easier.
#
com.ibm.ws.scriptingprofiles=/opt/IBM/WebSphere/AppServer/bin/securityProcs.py;
/opt/IBM/WebSphere/AppServer/bin/LTPA_LDAPSecurityProcs.py
Ln 101, Col 24 INS
```

These two Jython scripts run before any other command when wsadmin is started.



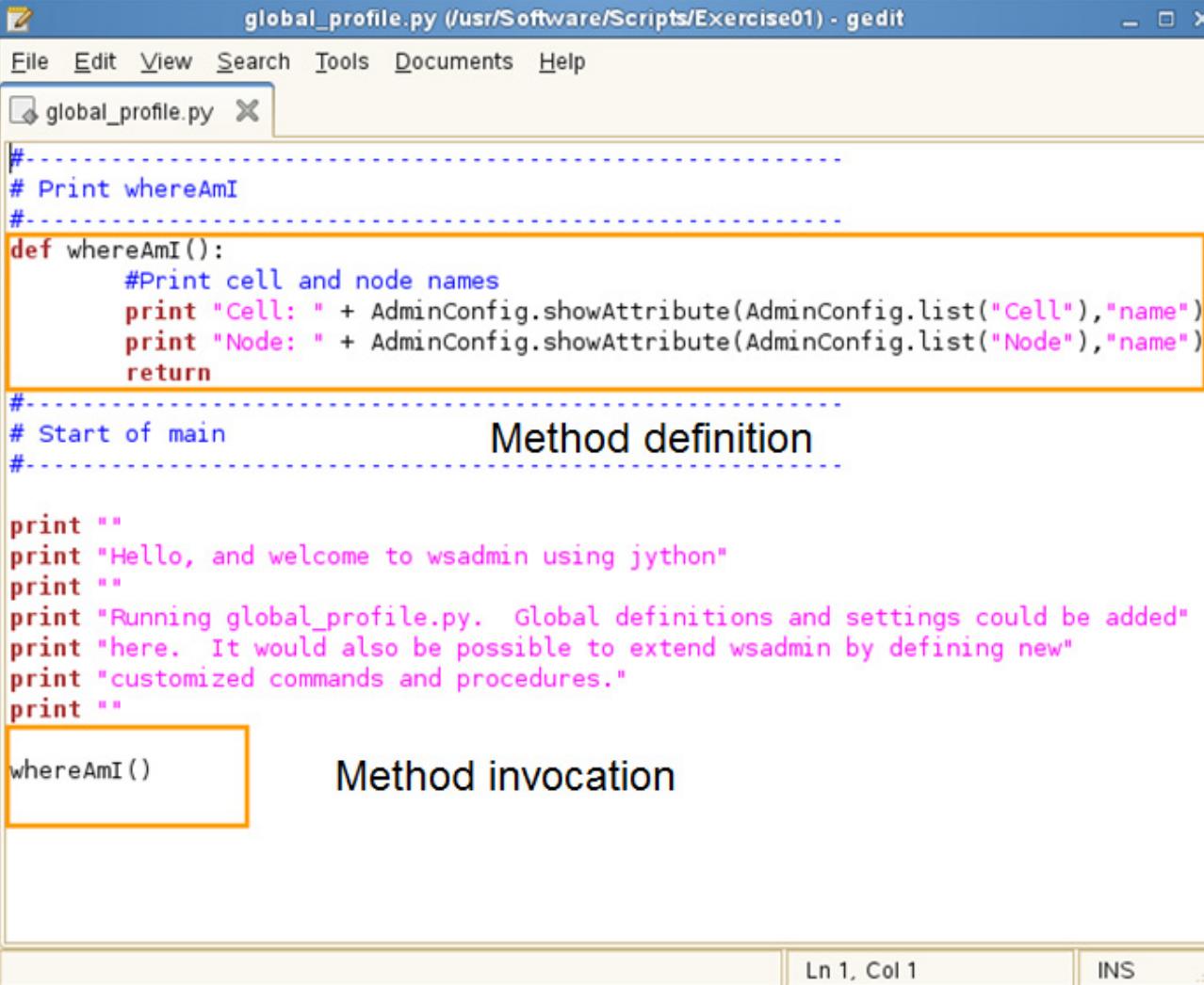
### Note

The `com.ibm.ws.scriptingprofiles` property can include several profile scripts. A semi-colon can be used to separate multiple scripts.

- \_\_ e. Save the changes.
- \_\_ f. Close the `wsadmin.properties` file.
- \_\_ 5. Profile files are wsadmin scripts that initialize variables and define methods for the mainline scripts that are run with the wsadmin tool. If multiple profile options are provided, they are run in the order listed.

Explore the `global_profile.py` script and then start the script by running the wsadmin tool with the `-profile` option and include the file name of the script.

- \_\_\_ a. Using a file system browser, go to the `/usr/Software/Scripts/Exercise01` directory.
- \_\_\_ b. Open the `global_profile.py` file by using a text editor.



```

global_profile.py (/usr/Software/Scripts/Exercise01) - gedit
File Edit View Search Tools Documents Help
global_profile.py X
#
# Print whereAmI
#
def whereAmI():
    #Print cell and node names
    print "Cell: " + AdminConfig.showAttribute(AdminConfig.list("Cell"), "name")
    print "Node: " + AdminConfig.showAttribute(AdminConfig.list("Node"), "name")
    return
#
# Start of main
#
print ""
print "Hello, and welcome to wsadmin using jython"
print ""
print "Running global_profile.py. Global definitions and settings could be added"
print "here. It would also be possible to extend wsadmin by defining new"
print "customized commands and procedures."
print ""

whereAmI()

```

**Method definition**

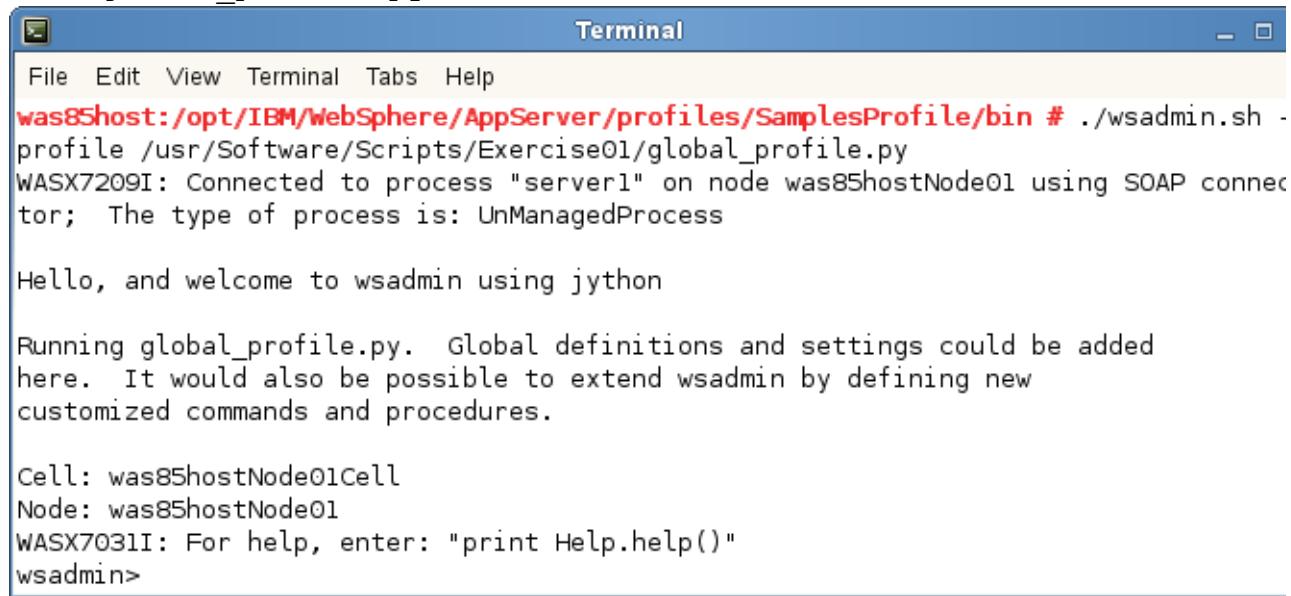
**Method invocation**

Ln 1, Col 1      INS

A method named `whereAmI()` is defined and uses the `AdminConfig` administrative object to retrieve the cell name and node name properties. The `whereAmI()` method is not started until the end of the main section. The main section prints information to the standard output.

- \_\_\_ c. Run the following command to start the `global_profile.py` script:

```
./wsadmin.sh -profile /usr/Software/Scripts/Exercise01/
global_profile.py
```



The screenshot shows a terminal window titled "Terminal". The session starts with the command `./wsadmin.sh -profile /usr/Software/Scripts/Exercise01/global_profile.py`. It then displays a welcome message: "Hello, and welcome to wsadmin using jython". A note follows: "Running global\_profile.py. Global definitions and settings could be added here. It would also be possible to extend wsadmin by defining new customized commands and procedures.". The environment variables are listed: "Cell: was85hostNode01Cell", "Node: was85hostNode01", and "WASX7031I: For help, enter: "print Help.help()"".

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./wsadmin.sh -profile /usr/Software/Scripts/Exercise01/global_profile.py
WASX7209I: Connected to process "server1" on node was85hostNode01 using SOAP connector; The type of process is: UnManagedProcess

Hello, and welcome to wsadmin using jython

Running global_profile.py. Global definitions and settings could be added here. It would also be possible to extend wsadmin by defining new customized commands and procedures.

Cell: was85hostNode01Cell
Node: was85hostNode01
WASX7031I: For help, enter: "print Help.help()"
wsadmin>
```

wsadmin is started in interactive mode and the `global_profile.py` script is run. Profile scripts are especially useful when you want to make variables and methods available for use by other scripts.

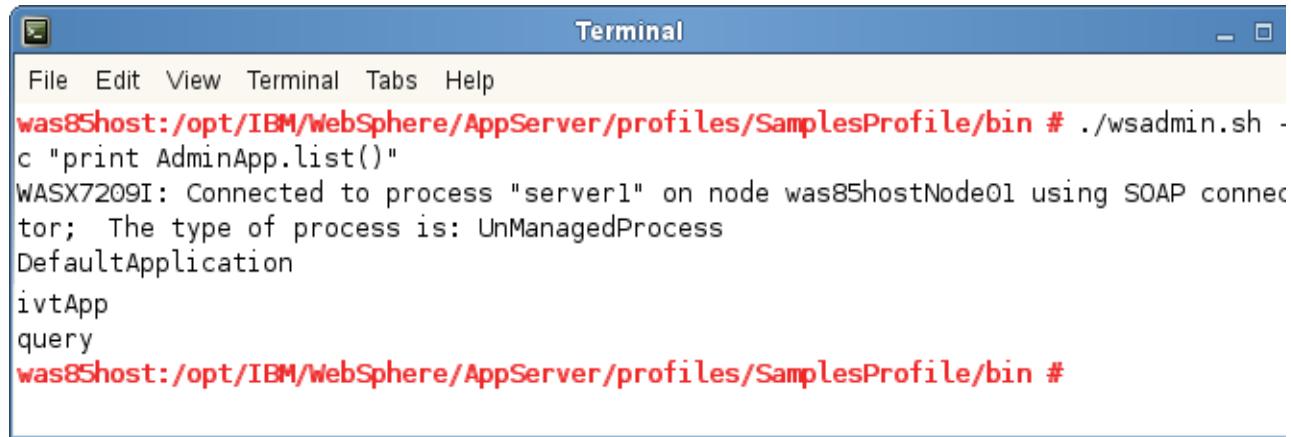
- \_\_\_ d. Exit wsadmin.

### **Section 3: Starting single Jython commands**

The ability to run single Jython scripts is available by using the `-c` option followed by a command when starting wsadmin.

- \_\_\_ 1. Run a single Jython command by using the `-c` wsadmin startup option.
- \_\_\_ a. Enter the following command to run a single command:

```
./wsadmin.sh -c "print AdminApp.list()"
```



The screenshot shows a terminal window titled "Terminal". The command `./wsadmin.sh -c "print AdminApp.list()"` is run. The output lists three enterprise applications: "DefaultApplication", "ivtApp", and "query". The session ends with the prompt `was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin #`.

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./wsadmin.sh -c "print AdminApp.list()"
WASX7209I: Connected to process "server1" on node was85hostNode01 using SOAP connector; The type of process is: UnManagedProcess
DefaultApplication
iwtApp
query
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin #
```

The command returns a list of the three installed enterprise applications.

**Important**

After running, the wsadmin shell is stopped. Running single scripts is not an efficient way to run scripts since a JVM must be created every time that a command is run.

## Section 4: Running Jython commands

There are five wsadmin administrative objects available: AdminConfig, AdminControl, AdminApp, AdminTask, and Help. Scripts use these objects for application management, configuration, operational control, and for communication with MBeans that run in WebSphere Application Server.

In this section, you run several scripts by using wsadmin in interactive mode.

- \_\_\_ 1. Start wsadmin in interactive mode.
- \_\_\_ a. Run the following command to start wsadmin in interactive mode:

```
./wsadmin.sh
```

```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./wsadmin.sh
WASX7209I: Connected to process "server1" on node was85hostNode01 using SOAP connector; The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>
```

You can now run any valid Jython commands and any help methods.

- \_\_\_ 2. Use the AdminControl object to get information about the cell, node, and host.
- \_\_\_ a. At the wsadmin command prompt, enter the following command:

```
print AdminControl.getCell()
```

```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./wsadmin.sh
WASX7209I: Connected to process "server1" on node was85hostNode01 using SOAP connector; The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>print AdminControl.getCell()
was85hostNode01Cell
wsadmin>
```

The cell name is displayed.

- \_\_\_ b. At the wsadmin command prompt, enter the following command:

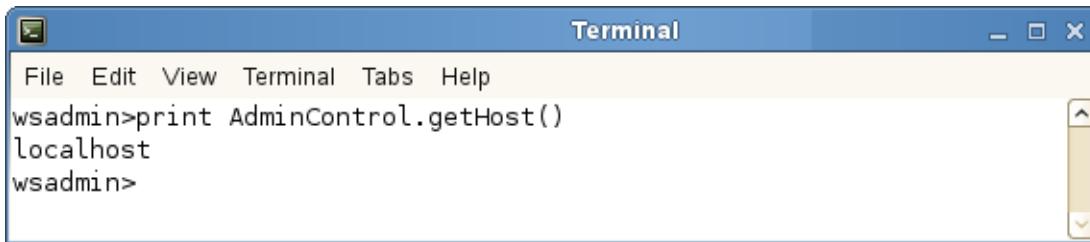
```
print AdminControl.getNode()
```

```
Terminal
File Edit View Terminal Tabs Help
wsadmin>print AdminControl.getNode()
was85hostNode01
wsadmin>
```

The node name is displayed.

- \_\_\_ c. At the wsadmin command prompt, enter the following command:

```
print AdminControl.getHost()
```



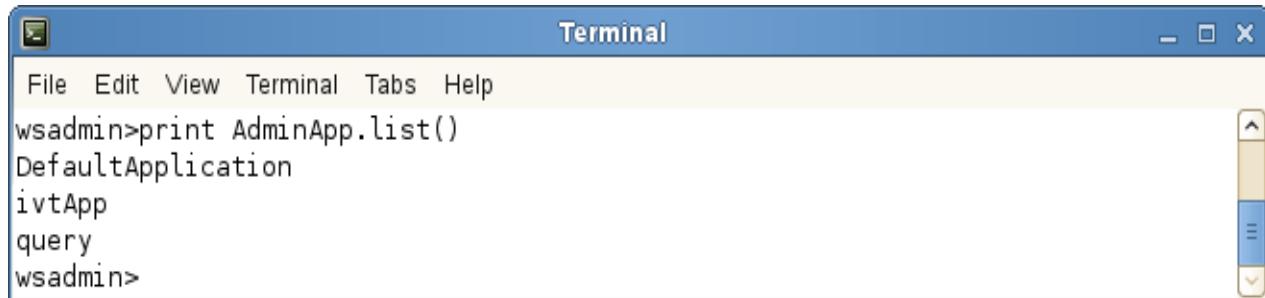
A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area shows the wsadmin command prompt. The user has entered the command `print AdminControl.getHost()`. The output is "localhost". The prompt then changes to "wsadmin>".

The host name is displayed.

- \_\_\_ 3. Use the AdminApp object to list applications and application module information.

- \_\_\_ a. At the wsadmin command prompt, enter the following command:

```
print AdminApp.list()
```



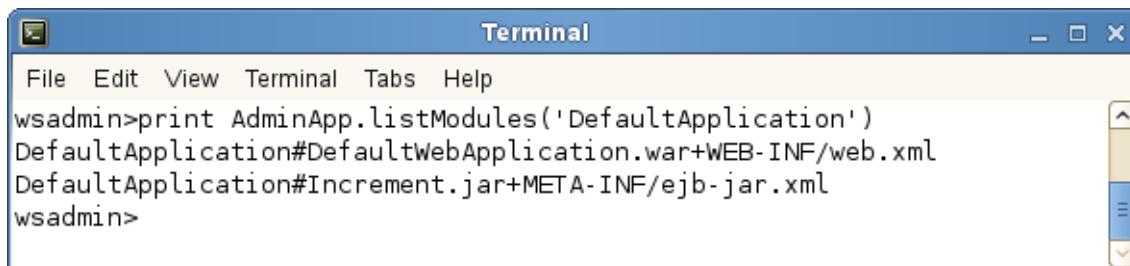
A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area shows the wsadmin command prompt. The user has entered the command `print AdminApp.list()`. The output lists two enterprise applications: "DefaultApplication" and "ivtApp". The prompt then changes to "wsadmin>".

A list of the installed enterprise applications is displayed.

- \_\_\_ 4. Use the AdminApp object to list the modules for a specified application.

- \_\_\_ a. At the wsadmin command prompt, enter the following command:

```
print AdminApp.listModules('DefaultApplication')
```



A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area shows the wsadmin command prompt. The user has entered the command `print AdminApp.listModules('DefaultApplication')`. The output lists the modules for the "DefaultApplication": "DefaultApplication#DefaultWebApplication.war+WEB-INF/web.xml" and "DefaultApplication#Increment.jar+META-INF/ejb-jar.xml". The prompt then changes to "wsadmin>".

A list of the installed web and EJB modules for the Default enterprise application is displayed.

- \_\_\_ 5. Exit the wsadmin interactive mode.

## Section 5: Loading and running Jython scripts

In this step, you run a script by using the wsadmin -f startup option. Although the interactive wsadmin interface is a powerful tool, it is much more efficient to create Jython scripts and then run them. Scripts help you reduce your editing and debugging time.

- \_\_\_ 1. Open the `exploreWebSphereApps.py` script.
  - \_\_\_ a. Using a file system browser, go to the `/usr/Software/Scripts/Exercise01` directory.
  - \_\_\_ b. Open the `exploreWebSphereApps.py` file by using a text editor.
- \_\_\_ 2. Examine the `exploreWebSphereApps.py` script.
  - \_\_\_ a. The Jython script runs the same commands that are listed in the previous steps but takes advantage of looping, array, and user input features of the Jython language.

```

exploreWebSphereApps.py (/usr/Software/Scripts/Exercise01) - gedit
File Edit View Search Tools Documents Help
exploreWebSphereApps.py X

print 'Cell name: ' + AdminControl.getCell()
print 'Node name: ' + AdminControl.getNode()
print 'Host name: ' + AdminControl.getHost()

print 'Installed applications:'

appList=AdminApp.list().split(lineSeparator)

for i in range (len(appList)):
    print i, appList[i]          for loop

print "\n"
inputValue= input( "Choose an application from above to list their modules: ")
print "\n"

print appList[inputValue], "Modules"        input
print "-----"
print AdminApp.listModules (appList[inputValue])   array

```

The screenshot shows a Gedit window with the title "exploreWebSphereApps.py (/usr/Software/Scripts/Exercise01) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The window title bar also has a close button. The code in the editor is highlighted in blue and pink. A yellow box highlights the for loop, another yellow box highlights the input statement, and a third yellow box highlights the print statement for AdminApp.listModules. The status bar at the bottom shows "Ln 1, Col 1" and "INS".

**Note**

Note the use of the variable **lineSeparator**. This variable is not defined anywhere in the script, yet it can be used. What makes this variable available to the script? In a previous section, you looked at the profiles that are loaded when wsadmin starts, these profiles were defined in the `wsadmin.properties` file. One of the two profiles is `LTPA_LDAPSecurity.py`. In this file, there is a line of code that defines the **lineSeparator** variable:

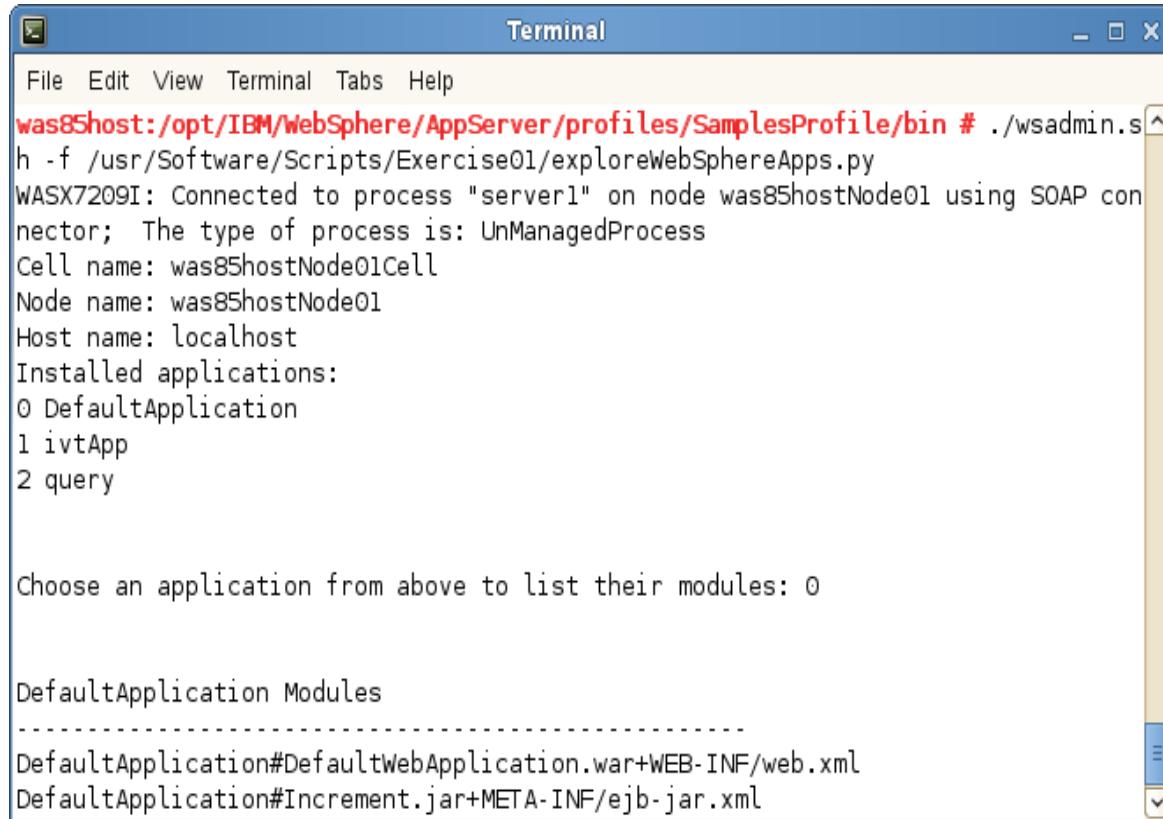
```
lineSeparator = java.lang.System.getProperty('line.separator')
```

Because the variable was defined in a profile, it becomes part of the environment and is available to other scripts.

It is generally not a good idea to depend on profile scripts that you do not explicitly load yourself because the writer or the owner of the properties file might modify them.

- \_\_\_ b. Close the `exploreWebSphereApps.py` file.
- \_\_\_ 3. Run the `exploreWebSphereApps.py` script by using the wsadmin -f option.
  - \_\_\_ a. In a terminal window, go to the `<profile_root>/SamplesProfile/bin` directory.
  - \_\_\_ b. Enter the following command to run the `exploreWebSphereApps.py` script:  
`./wsadmin.sh -f /usr/Software/Scripts/Exercise01/exploreWebSphereApps.py`

- \_\_\_ c. When the script prompts for user input, enter the value **0** and press the Enter key:



```

Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./wsadmin.sh -f /usr/Software/Scripts/Exercise01/exploreWebSphereApps.py
WASX7209I: Connected to process "server1" on node was85hostNode01 using SOAP connector; The type of process is: UnManagedProcess
Cell name: was85hostNode01Cell
Node name: was85hostNode01
Host name: localhost
Installed applications:
0 DefaultApplication
1 ivtApp
2 query

Choose an application from above to list their modules: 0

DefaultApplication Modules
-----
DefaultApplication#DefaultWebApplication.war+WEB-INF/web.xml
DefaultApplication#Increment.jar+META-INF/ejb-jar.xml

```

The modules for the DefaultApplication application are displayed. The script completes, and wsadmin exits.



### Important

After the script file runs, the wsadmin session is stopped. When developing, testing, and debugging using text editors, this method is not an efficient way to run scripts since a JVM must be created every time that a script is run.

- \_\_\_ 4. Modify the `exploreWebSphereApp.py` script and use the `execfile()` method from the wsadmin interactive mode to call the script.
- \_\_\_ a. Using a file system browser, go to the `/usr/Software/Scripts/Exercise01` directory.
  - \_\_\_ b. Open the `exploreWebSphereApps.py` file by using a text editor.
  - \_\_\_ c. Locate the line that reads:
- ```
print i, appList[i]
```

- \_\_\_ d. Update the line as follows to add parentheses around the count value:

```
print '(',i,')',appList[i]
```

```
exploreWebSphereApps.py (/usr/Software/Scripts/Exercise01) - gedit
File Edit View Search Tools Documents Help
exploreWebSphereApps.py X

print 'Cell name: ' + AdminControl.getCell()
print 'Node name: ' + AdminControl.getNode()
print 'Host name: ' + AdminControl.getHost()

print 'Installed applications:'

appList=AdminApp.list().split(lineSeparator)

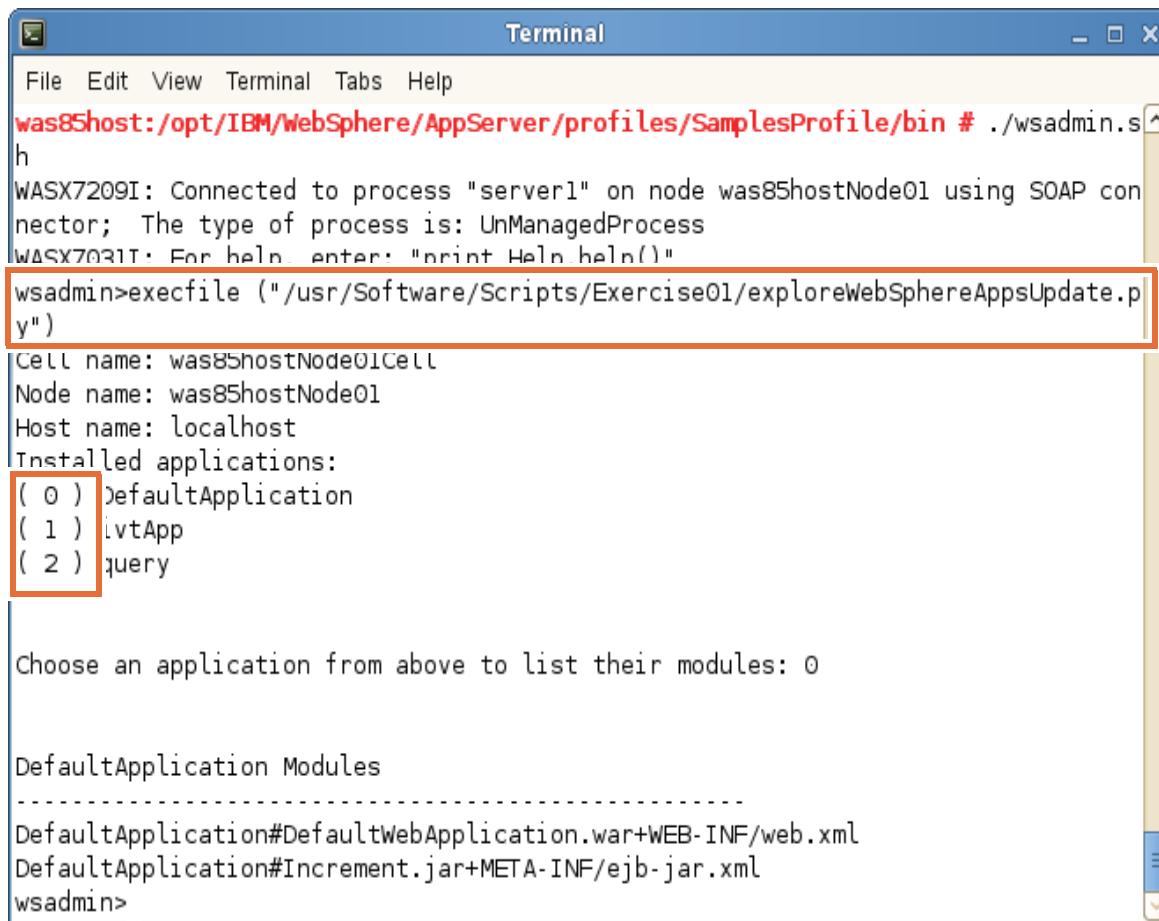
for i in range (len(appList)):
    print '(',i,')', appList[i]

print "\n"
inputValue= input( "Choose an application from above to list their modules: ")
print "\n"

print appList[inputValue], "Modules"
print "-----"
print AdminApp.listModules (appList[inputValue])|
```

- \_\_\_ e. Save the file as /usr/Software/Scripts/Exercise01/exploreWebSphereAppsUpdate.py
- \_\_\_ f. Close the **exploreWebSphereAppsUpdate.py** file.
- \_\_\_ g. Start wsadmin in interactive mode by entering the following command in the command prompt:  
**./wsadmin.sh**
- \_\_\_ h. Enter the following command to run the updated script file and remember to enter the value **0** when prompted:

```
execfile ("/usr/Software/Scripts/Exercise01/  
exploreWebSphereAppsUpdate.py")
```



```
Terminal  
File Edit View Terminal Tabs Help  
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./wsadmin.s  
h  
WASX7209I: Connected to process "server1" on node was85hostNode01 using SOAP con  
nector; The type of process is: UnManagedProcess  
WASX7031I: For help enter: "print Help.help()  
wsadmin>execfile ("/usr/Software/Scripts/Exercise01/exploreWebSphereAppsUpdate.p  
y")  
Cell name: was85hostNode01Cell  
Node name: was85hostNode01  
Host name: localhost  
Installed applications:  
( 0 ) DefaultApplication  
( 1 ) ivtApp  
( 2 ) query  
  
Choose an application from above to list their modules: 0  
  
DefaultApplication Modules  
-----  
DefaultApplication#DefaultWebApplication.war+WEB-INF/web.xml  
DefaultApplication#Increment.jar+META-INF/ejb-jar.xml  
wsadmin>
```

The count value is updated to display parentheses. Using the `execfile()` method to run Jython scripts does not require you to start a new `wsadmin` session each time a script is run. This method is useful when writing, debugging, and testing Jython scripts.

- \_\_\_ i. Exit `wsadmin` by typing **quit**.
- \_\_\_ 5. Stop WebSphere Application Server.
  - \_\_\_ a. Open a terminal window by clicking the **GNOME Terminal** icon.
  - \_\_\_ b. Go to the `<profile_root>/SamplesProfile/bin` directory.
  - \_\_\_ c. Enter the following command to stop the server1:  
`./stopServer.sh server1 -username wasadmin -password  
web1sphere`

## Section 6: Optional: Encoding passwords in a property file

The purpose of password encoding is to deter casual observation of passwords in server configuration and property files. WebSphere Application Server uses the `PropFilePasswordEncoder` utility to encode any passwords and properties and to remove all comments from the production versions of property files. By default, the `PropFilePasswordEncoder` utility encodes all passwords by using an XOR mask.

To encode the properties of a property file, use the following command format:

```
<was_root>/bin/PropFilePasswordEncoder filename property_name
```

Where:

- **filename** is the name of the target properties file for password encoding.
- **property\_name** is the name of the specific property to encode.



### Warning

If no property name is specified, all properties in the file are encoded.



### Important

If a password is forgotten or it cannot be decoded, edit the file and set the password to the appropriate clear text value. Encode the property file after you change that value.



### Note

Create a backup copy before changing property and configuration files.

1. Encode the plain text password in the `soap.client.props` property file.
  - a. Using a command prompt, go to the `<profile_root>/SamplesProfile/bin` directory
  - b. Run the following command to encode the `com.ibm.SOAP.loginPassword` property:

```
./PropFilePasswordEncoder.sh
"/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/properties/soap.client.props" com.ibm.SOAP.loginPassword
```

```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin # ./PropFilePasswordEncoder.sh "/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/properties/soap.client.props" com.ibm.SOAP.loginPassword
was85host:/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/bin #
```

The PropFilePasswordEncoder utility completed successfully. The soap.client.props file now has XOR encoded the com.ibm.SOAP.loginPassword property and removed all comments.

- 2. Review the changes to the soap.client.props file.
  - a. Using a file system browser, go to the <profile\_root>/SamplesProfile/properties directory.
  - b. Open the soap.client.props file by using a text editor to view the updates.

```
soap.client.props (/opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/properties) - s - □ x
File Edit View Search Tools Documents Help
soap.client.props X

#-----
# - authenticationTarget ( BasicAuth[default], KRB5.
# These are the only supported selection
# on a pure client for JMX SOAP Connector Client. )
#-----
com.ibm.SOAP.authenticationTarget=BasicAuth

com.ibm.SOAP.loginUserId=wasadmin
com.ibm.SOAP.loginPassword={xor}KDo9biwvNzot0g==
```

The com.ibm.SOAP.loginPassword property was XOR encoded and all the comments are removed except for the time stamp, which lists when the file was updated.

- c. Close the soap.client.props file.

**End of exercise**

## Exercise review and wrap-up

In this exercise, you started wsadmin by using the NONE connection type as well as single command, interactive, and script modes. You also reviewed and updated properties that wsadmin uses, explored the use of wsadmin profiles, and used the help utilities. You then updated a Jython script and ran it using the `-f` wsadmin option and the `execfile()` method.

Optionally, you used the `PropFilePasswordEncoder` utility to XOR encode any passwords and properties in a property file to deter casual observation.



# Exercise 2. Using the IBM Assembly and Deploy Tools (IADT) to develop Jython scripts

## What this exercise is about

In this exercise, you learn the basic constructs of the Jython language and how to create and run Jython scripts by using the IBM Assembly and Deploy Tools (IADT). In addition, you use the integrated Jython debugger to debug your scripts and the administrative console command assistance tool to help automate Jython script development.

## What you should be able to do

At the end of this exercise, you should be able to:

- Create, debug, and run Jython scripts in the IADT
- Use the development tool aids provided by the IADT
- Explore other functions of the Administrative objects

## Introduction

The IBM Assembly and Deploy Tools provides a rich set of tools that facilitate the development of Jython scripts. They include a:

- Jython editor, complete with color syntax highlights, code completion, and keyword help for wsadmin administrative objects.
- Script launcher so that you can run script files directly from the workspace on a target WebSphere Application Server.
- Debugger that enables you to detect and diagnose errors in your Jython scripts. It allows you to control the execution of your scripts by setting breakpoints, suspending threads, stepping through the code, and examining the contents of variables.
- Listener facility that captures command notifications that the administrative console command assistance tool sends. These wsadmin commands can then be easily imported into a script opened in the Jython editor, enabling you to develop Jython scripts that are based on actual console actions.

In this exercise, you familiarize yourself with the Jython language and the Jython script development environment that IADT provides by following these steps.

1. Open a new workspace, create a Jython project, and define a server
2. Create and run a “Hello world” Jython script
3. Explore Jython language constructs
4. Develop an administrative script
5. Debug a Jython script
6. Use the administrative console command assistance to help develop a Jython script

## **Requirements**

To complete this exercise, the WebSphere Application Server Network Deployment V8.5 product must be installed and a server named server1 must be created in the default profile. You also require the IBM Assembly and Deploy Tools for WebSphere Administration (IADT).

## Exercise instructions



### Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

`<was_root>`: /opt/IBM/WebSphere/AppServer

`<profile_root>`: /opt/IBM/WebSphere/AppServer/profiles

### **Section 1: Opening a new workspace, creating a Jython project and defining a server**

In this section, you open a new workspace in the IBM Assembly and Deploy Tools (IADT) and explore its help facility. You then create a Jython project to hold the scripts that you develop. You also import the utility script, `scriptExecutor.py`, which is used to facilitate script execution inside the IADT. Finally, you create a server definition for `server1` in `SamplesProfile` so that you can control the server directly from inside the tool.

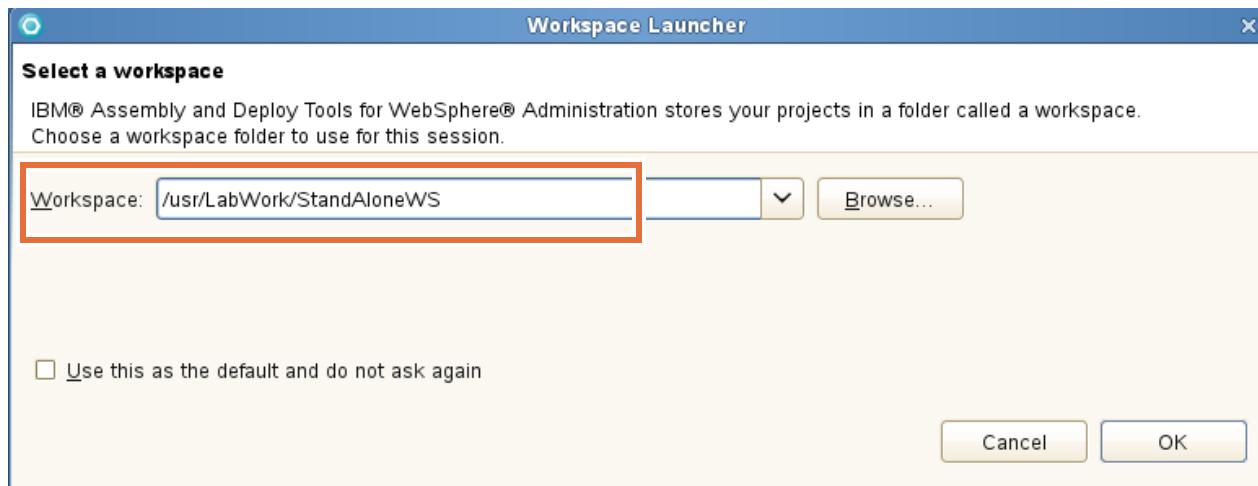
- 1. Start the IBM Assembly and Deploy Tools and open a new workspace.
  - a. On the desktop, double-click the **IBM Assembly and Deploy Tools** icon. The Workspace Launcher window opens and asks you to select a workspace.



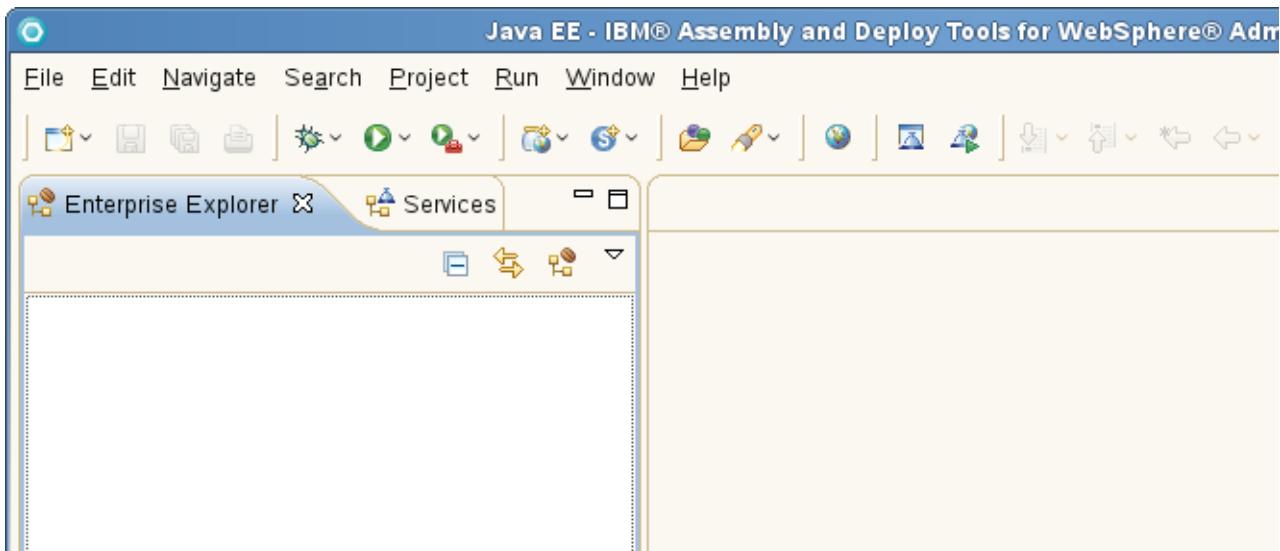
### Information

You can also start the IBM Assembly and Deploy Tools by selecting **Computer > More Applications**. Under Development, click **IBM IBM Assembly and Deploy Tools**.

- \_\_\_ b. A workspace specifies a location in the file system where the artifacts that you create are stored. In the Workspace Launcher window, type **/usr/LabWork/StandAloneWS** in the Workspace field.



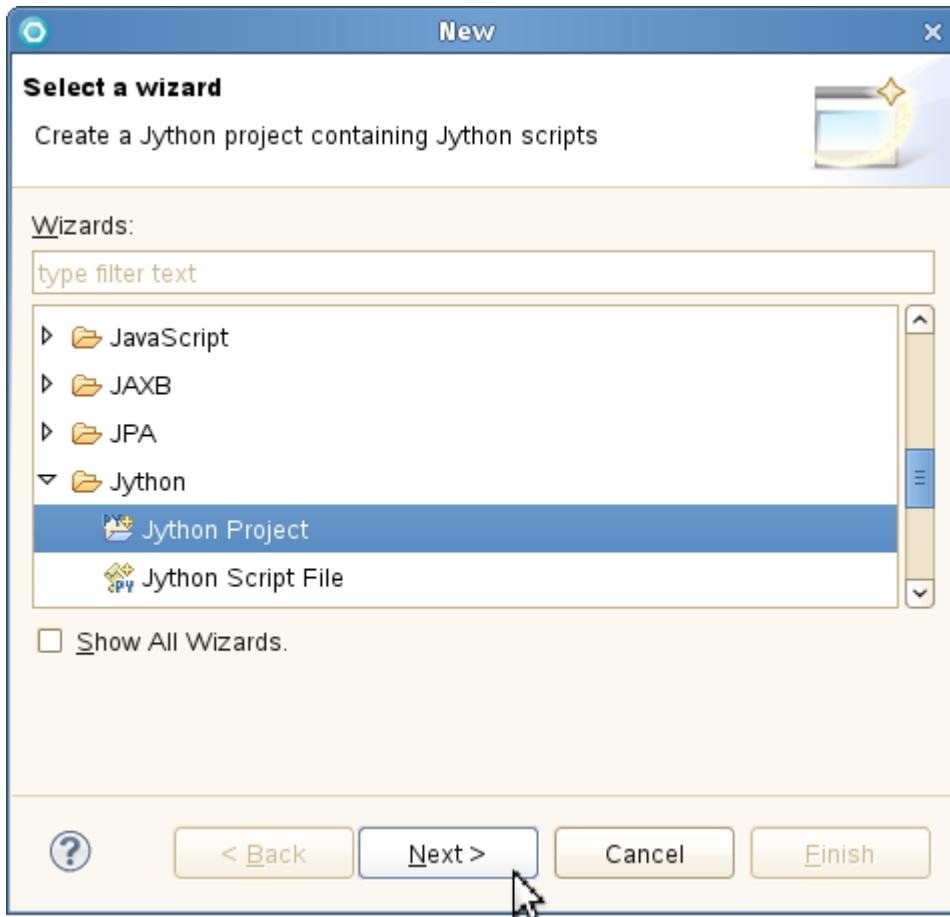
- \_\_\_ c. Click **OK**. The IADT opens and displays the Java EE perspective by default.



2. First, create a Jython project called **AdminScriptingProject**. A project is a folder that is used to contain a group of related development artifacts, Jython scripts in your case.
- a. From the main menu, select **File > New > Project**.

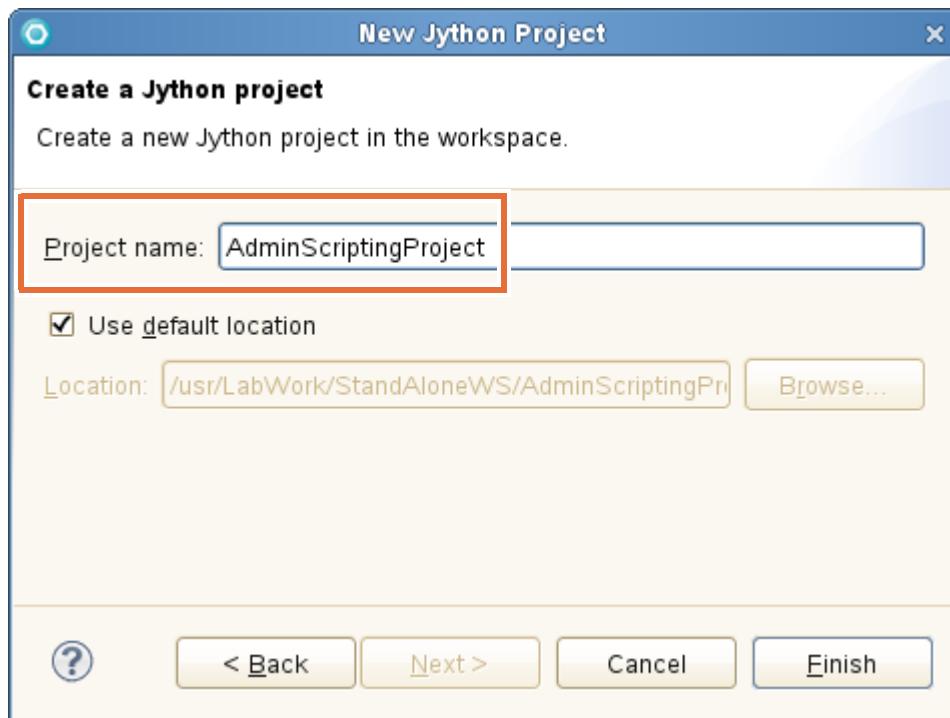


- \_\_\_ b. In the **Select a wizard** dialog, scroll down, expand **Jython**, and select **Jython Project**.

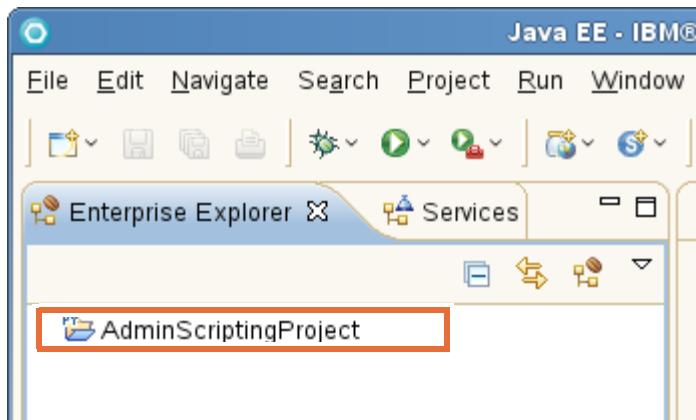


- \_\_\_ c. Click **Next**.

- \_\_\_ d. In the **Create a Jython project** dialog, type **AdminScriptingProject** in the Project name field.

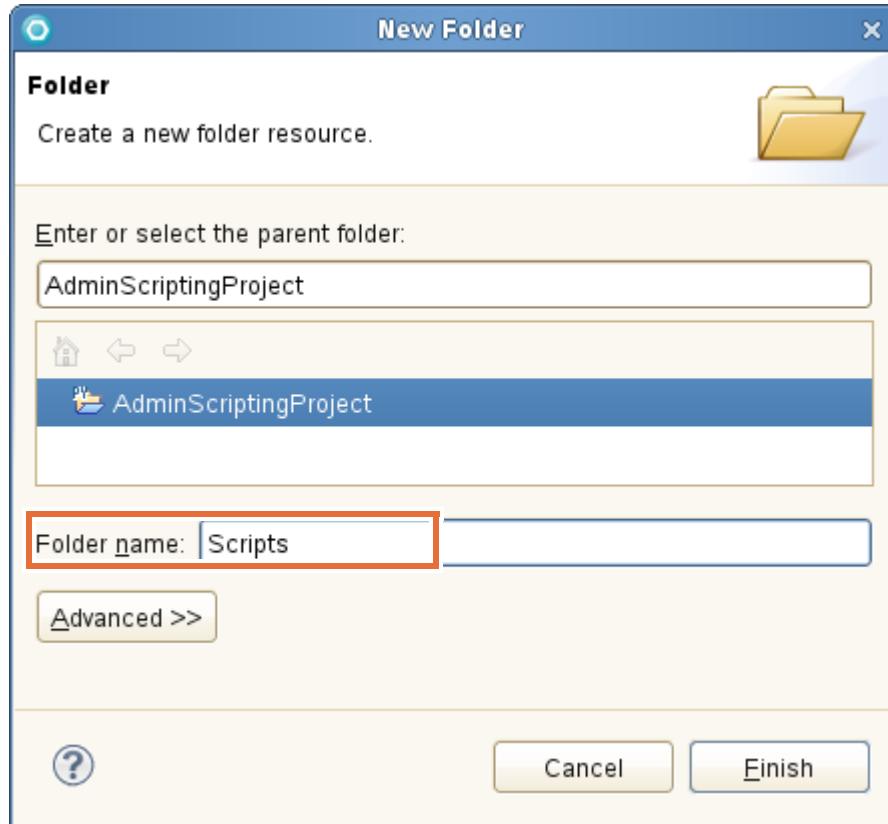


- \_\_\_ e. Click **Finish**. The AdminScriptingProject appears in the Enterprise Explorer view.

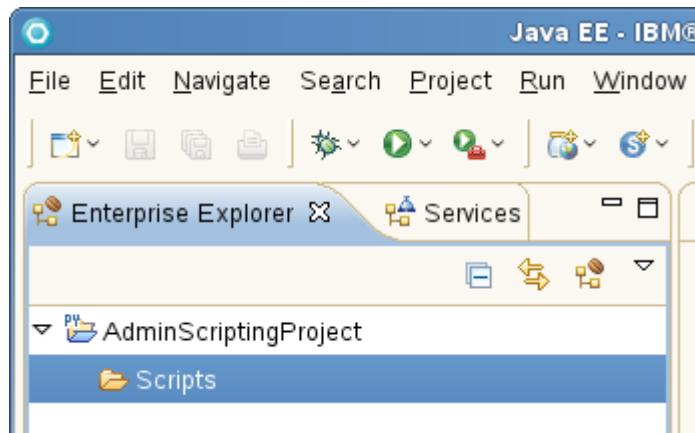


- \_\_\_ 3. Next, create a **Scripts** folder to hold your scripts.
- \_\_\_ a. In the Enterprise Explorer view, right-click **AdminScriptingProject** and select **New > Folder**.

- \_\_\_ b. In the **New Folder** dialog, type **Scripts** in the Folder name field.

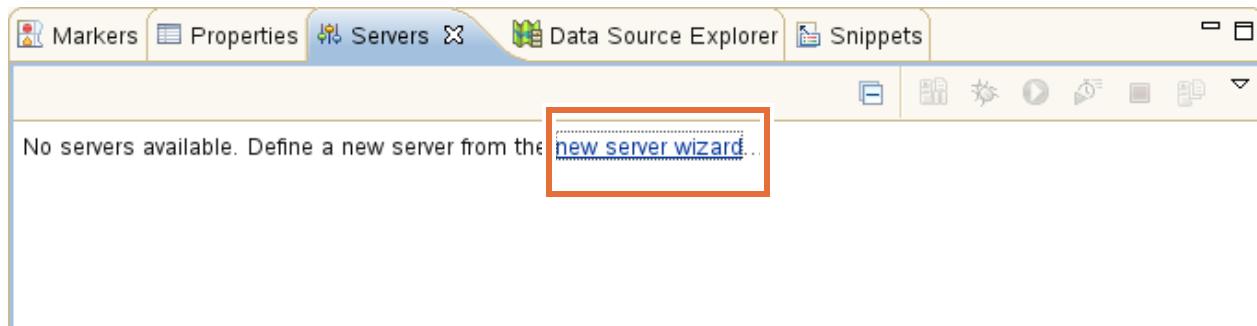


- \_\_\_ c. Click **Finish**. The new folder appears under the project in the Enterprise Explorer view.

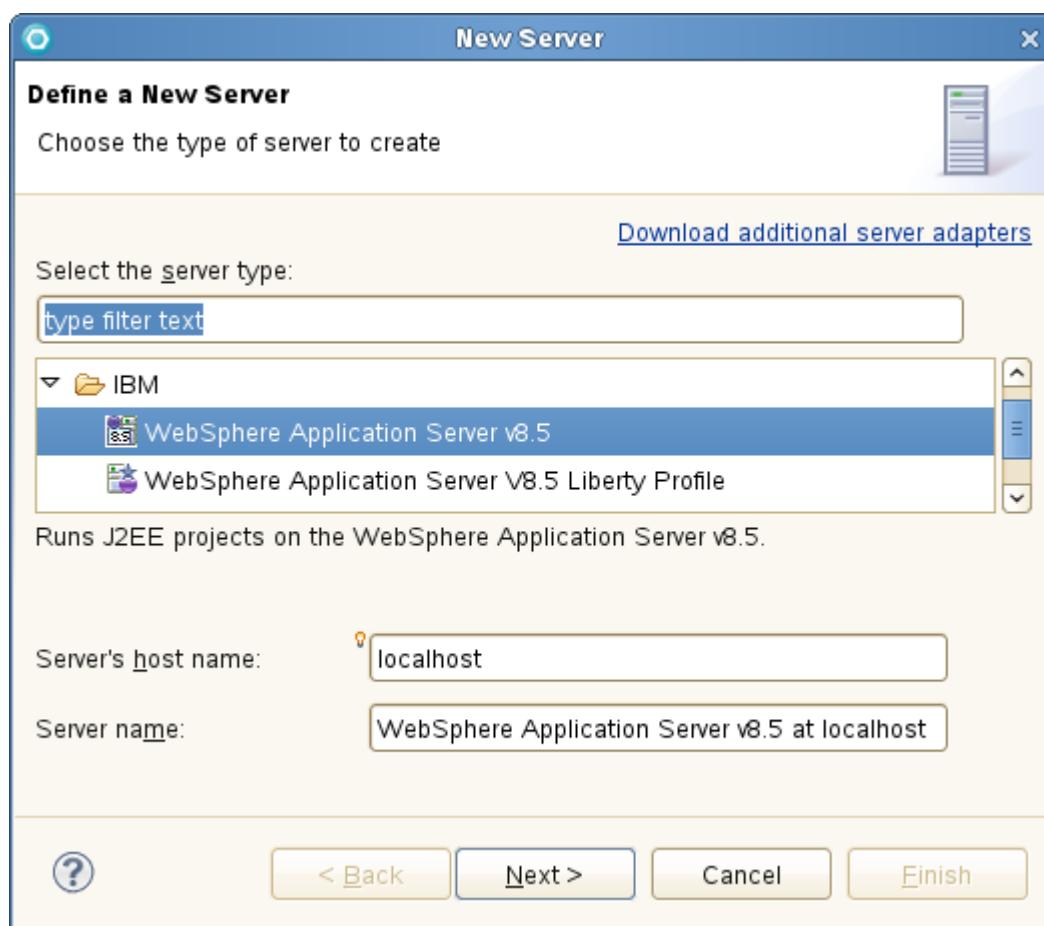


- \_\_\_ 4. Finally, create a server definition for **server1** in **SamplesProfile** and start the server.  
\_\_\_ a. Click the **Servers** tab to gain focus on the Servers view.

- \_\_ b. Click the link, **new server wizard**. Alternatively, right-click anywhere in the Servers view and select **New > Server**.

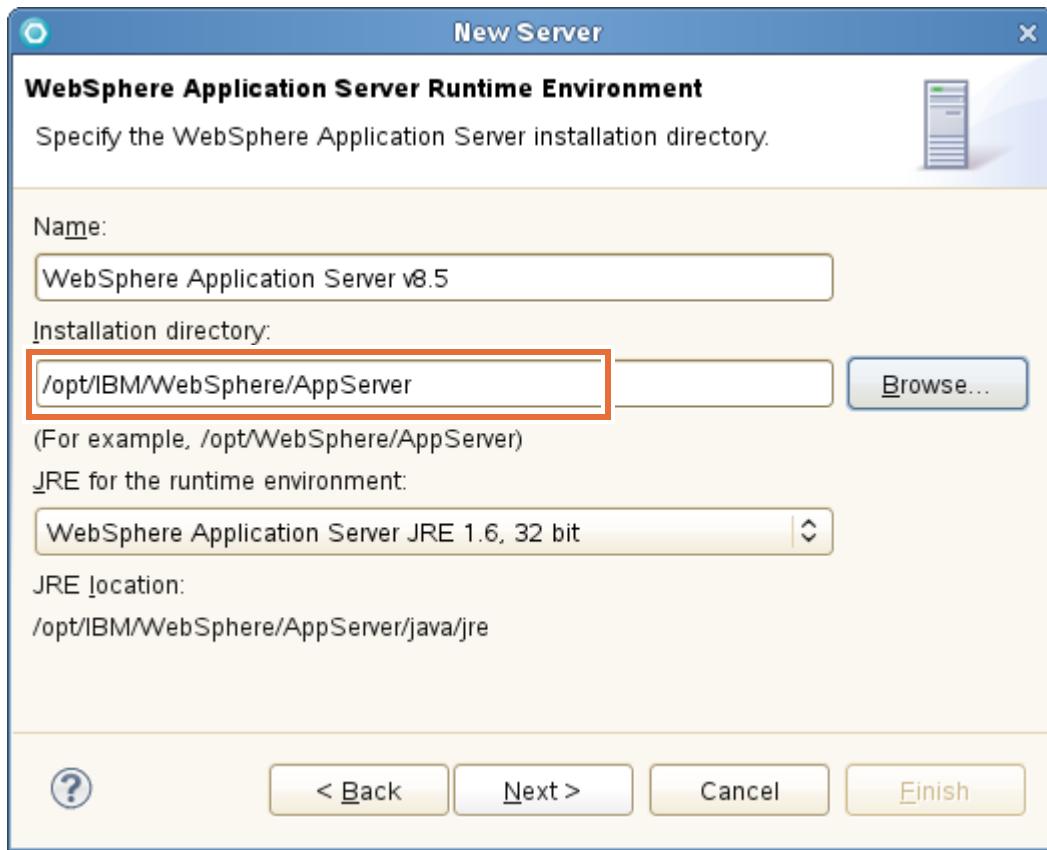


- \_\_ c. In the **Define a New server** dialog, scroll down, expand IBM, and select **WebSphere Application Server v8.5 Server**.

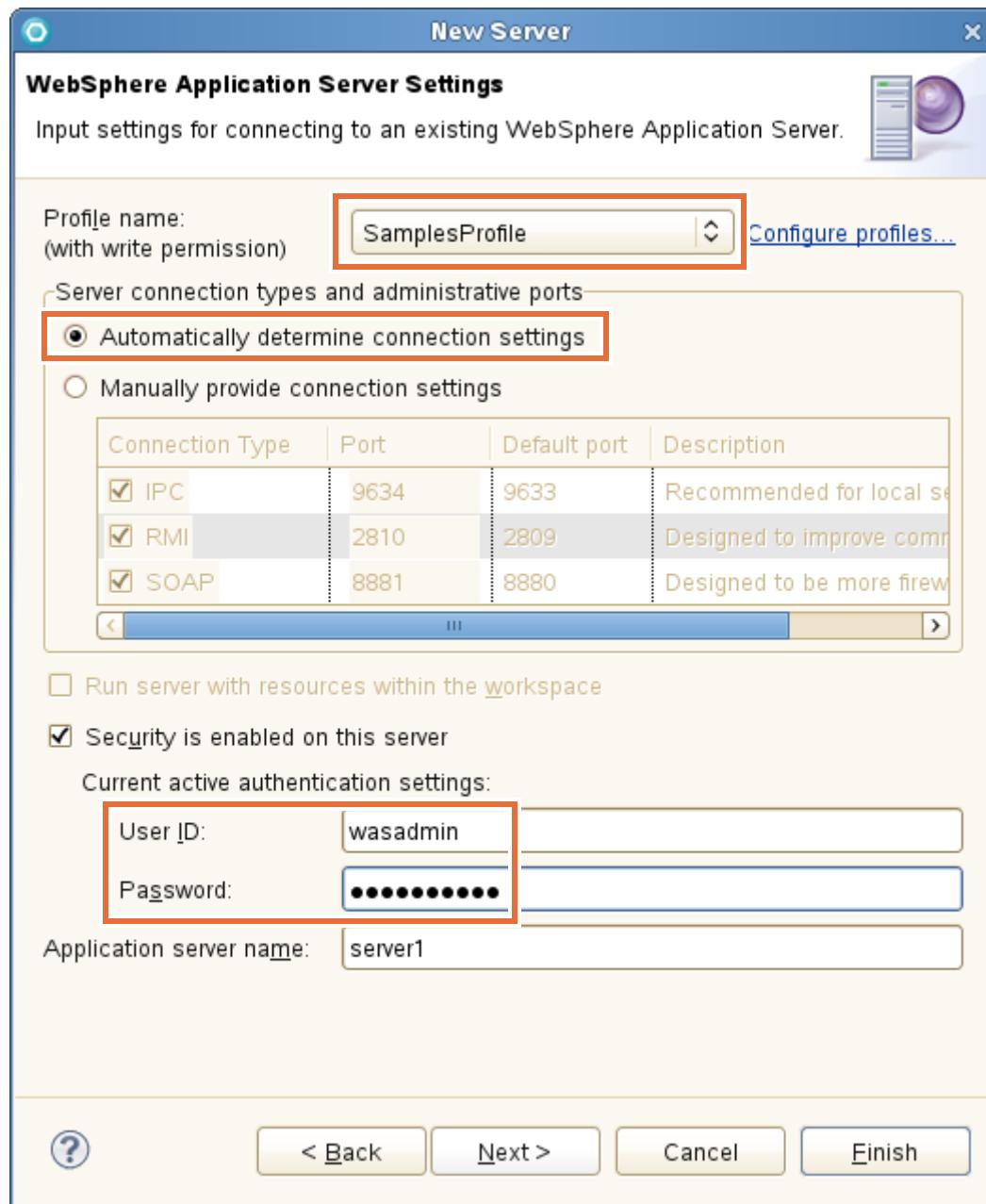


- d. Click **Next**.

- \_\_\_ e. In the **WebSphere Application Server Runtime** dialog, click **Browse** to browse for the installation directory field, or type `/opt/IBM/WebSphere/AppServer`.

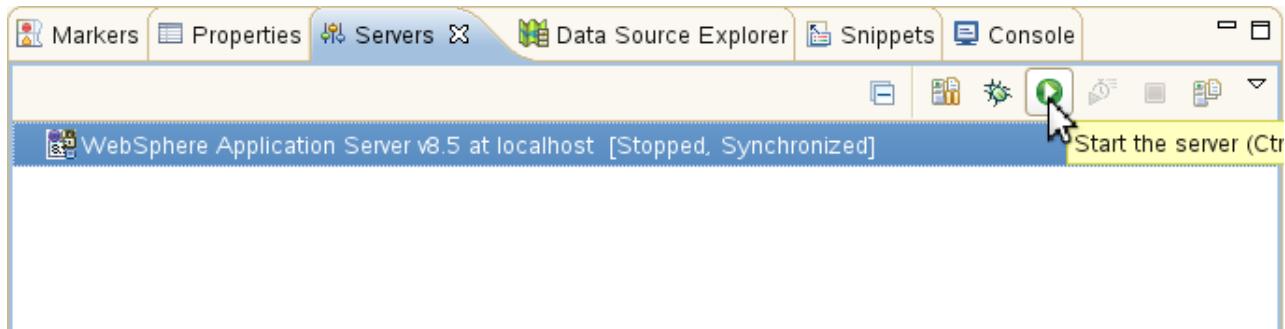


- \_\_\_ f. Click **Next**.
- \_\_\_ g. In the **WebSphere Server Settings** dialog:
- Make sure **SamplesProfile** is selected for Profile name.
  - Select the **Automatically determine connection settings**.
  - Since administrative security is enabled on the server, keep the **User ID** as `wasadmin`, and enter `web1sphere` in the **Password** field.



- Click **Finish**. After a moment, the SamplesProfile server1 is displayed in the Servers view as **WebSphere Application Server v8.5 at localhost**.

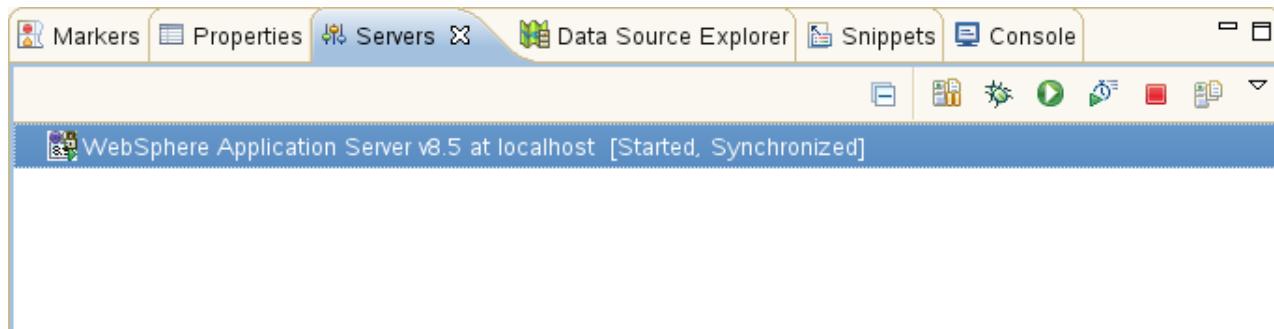
- \_\_ h. Start the SamplesProfile server1. In the Servers view, make sure **WebSphere Application Server v8.5 at localhost** is selected; then click **Start the server**.



- \_\_ i. Monitor the startup of the server. As the server starts, focus is switched to the **Console** view where startup messages are displayed.



- \_\_ j. Wait until you see the message: **server1 open for e-business** in the Console view.
- \_\_ k. When the server is started, click the **Servers** tab. The Servers view displays a status of **Started**.

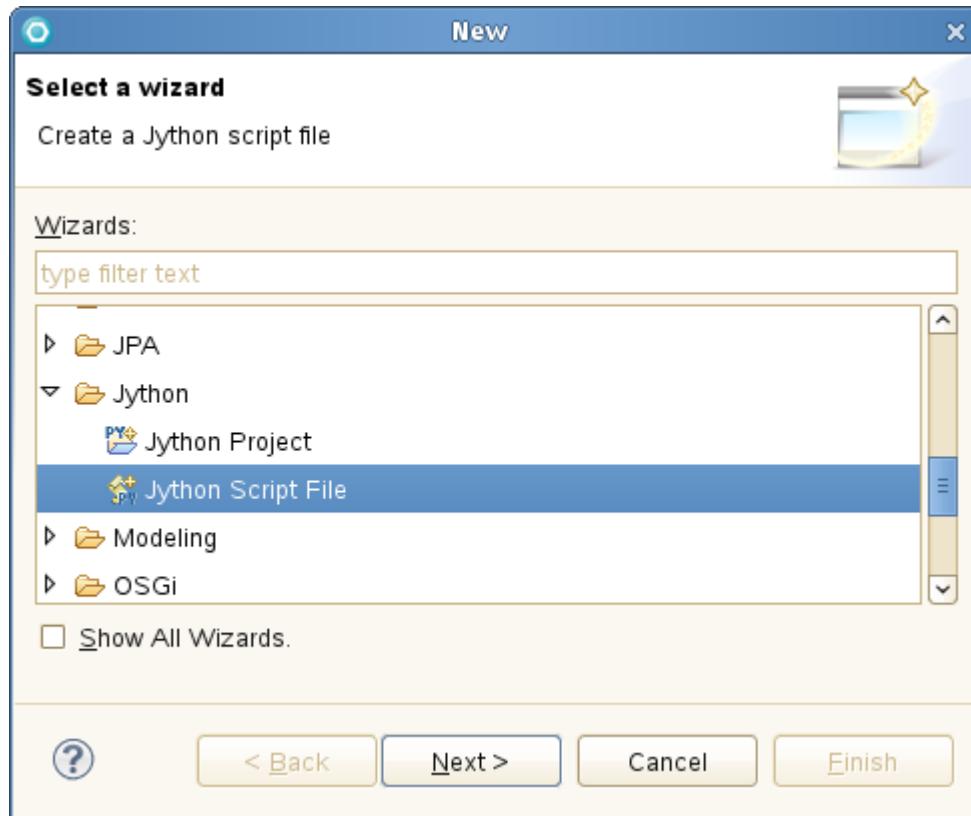


You successfully opened a new workspace, created a Jython project, and defined a server in the IBM Assembly and Deploy Tools.

## Section 2: Creating and running a “Hello world” Jython script

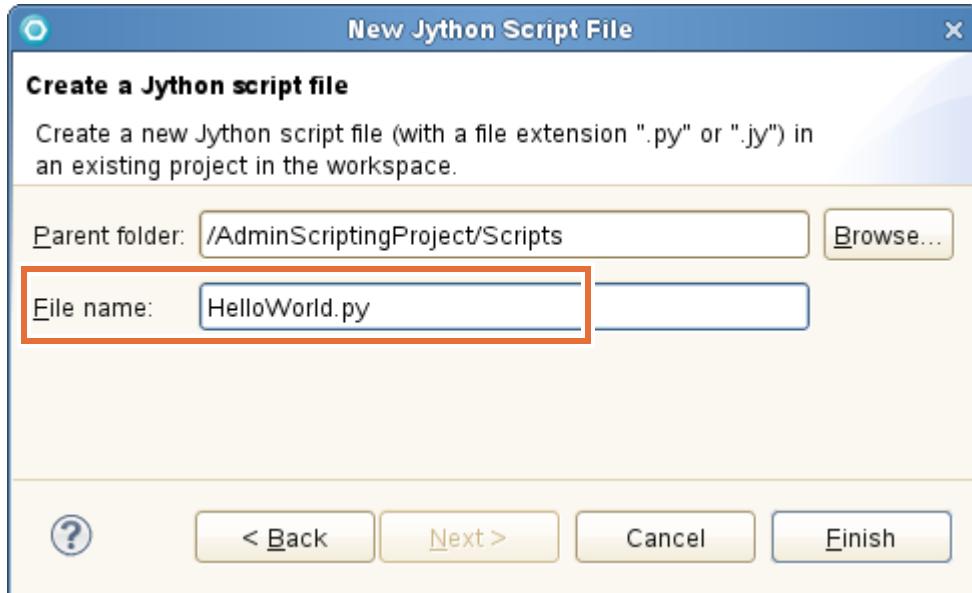
In this section, you begin your exploration of the Jython language and the IADT Jython development environment by creating and running a simple “Hello World” script.

- \_\_\_ 1. Create a `HelloWorld.py` script file by using the Jython Script File creation wizard.
  - \_\_\_ a. In the Enterprise Explorer view, right-click **Scripts** and select **New > Other**. The **Select a wizard** dialog opens.
  - \_\_\_ b. In the **Select a wizard** dialog, expand **Jython** and select **Jython Script File**.



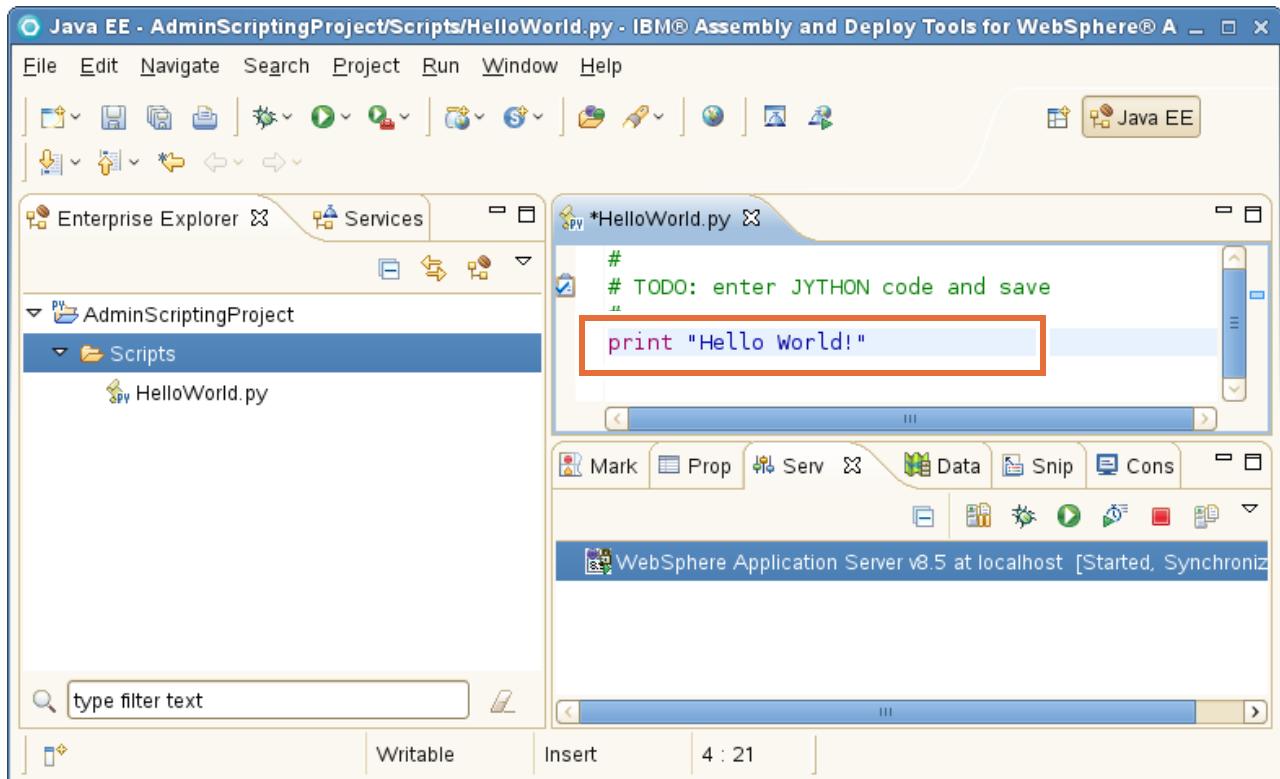
- \_\_\_ c. Click **Next**.

- \_\_\_ d. In the **Create a Jython script file** dialog, type **HelloWorld.py** in the File name field.



- \_\_\_ e. Click **Finish**. The script named **HelloWorld.py** now appears in the Enterprise Explorer view in the Scripts folder, and is opened in a Jython editor view. By default, it contains only comments lines, including the generic (and obvious) reminder: TODO: enter JYTHON code and save.
- \_\_\_ 2. Enter Jython code in the **HelloWorld** script file.
- \_\_\_ a. Add the following line of code in the Jython editor to display a greeting message:

```
print "Hello World!"
```



\_\_ b. Type **Ctrl+S** or select **File > Save** to save the change.



### Information

The Jython editor has many text-editing features, such as syntax-highlighting, unlimited undo or redo, and automatic tab indentation. When tagging a comment in a Jython script with `#TODO`, the editor automatically creates a corresponding task as a reminder in the **Tasks** view. Later, if you select that task in the **Tasks** view, the script file that contains it is automatically opened in an editor with the cursor positioned on the line that contains the corresponding `TODO` entry.

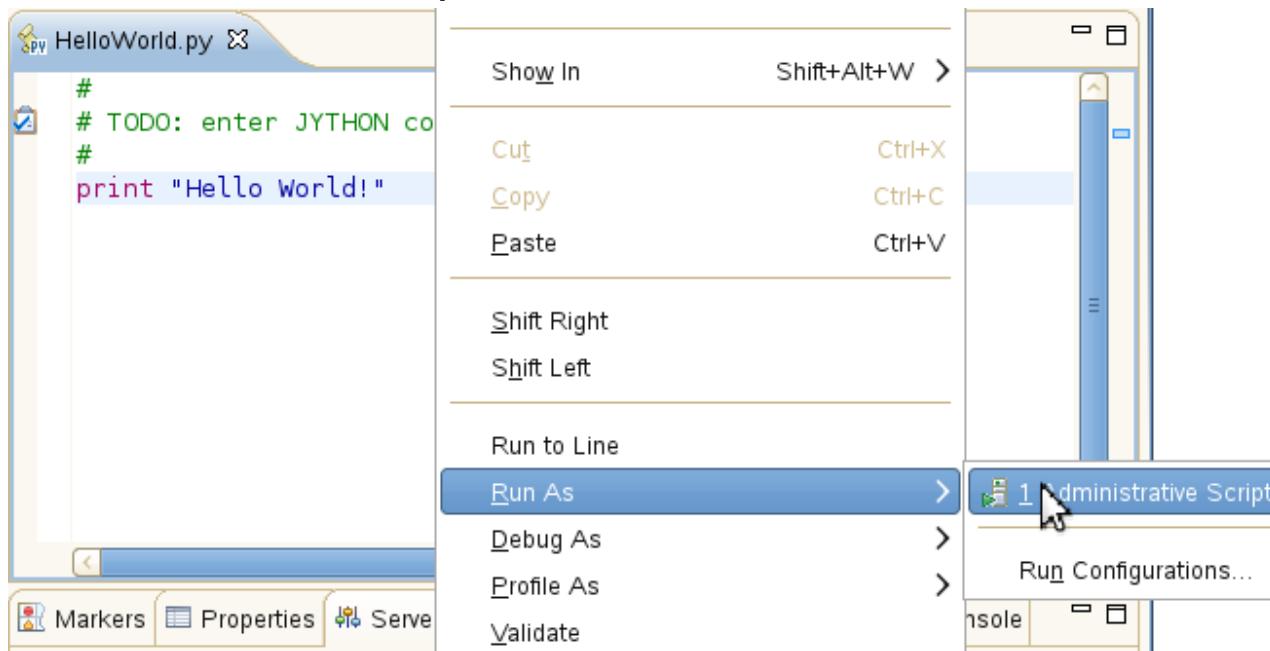
Other helpful features are content assist and content tips, which provides a list of acceptable continuations that depend on where the cursor is located in a Jython script file, or what you recently typed.



### Warning

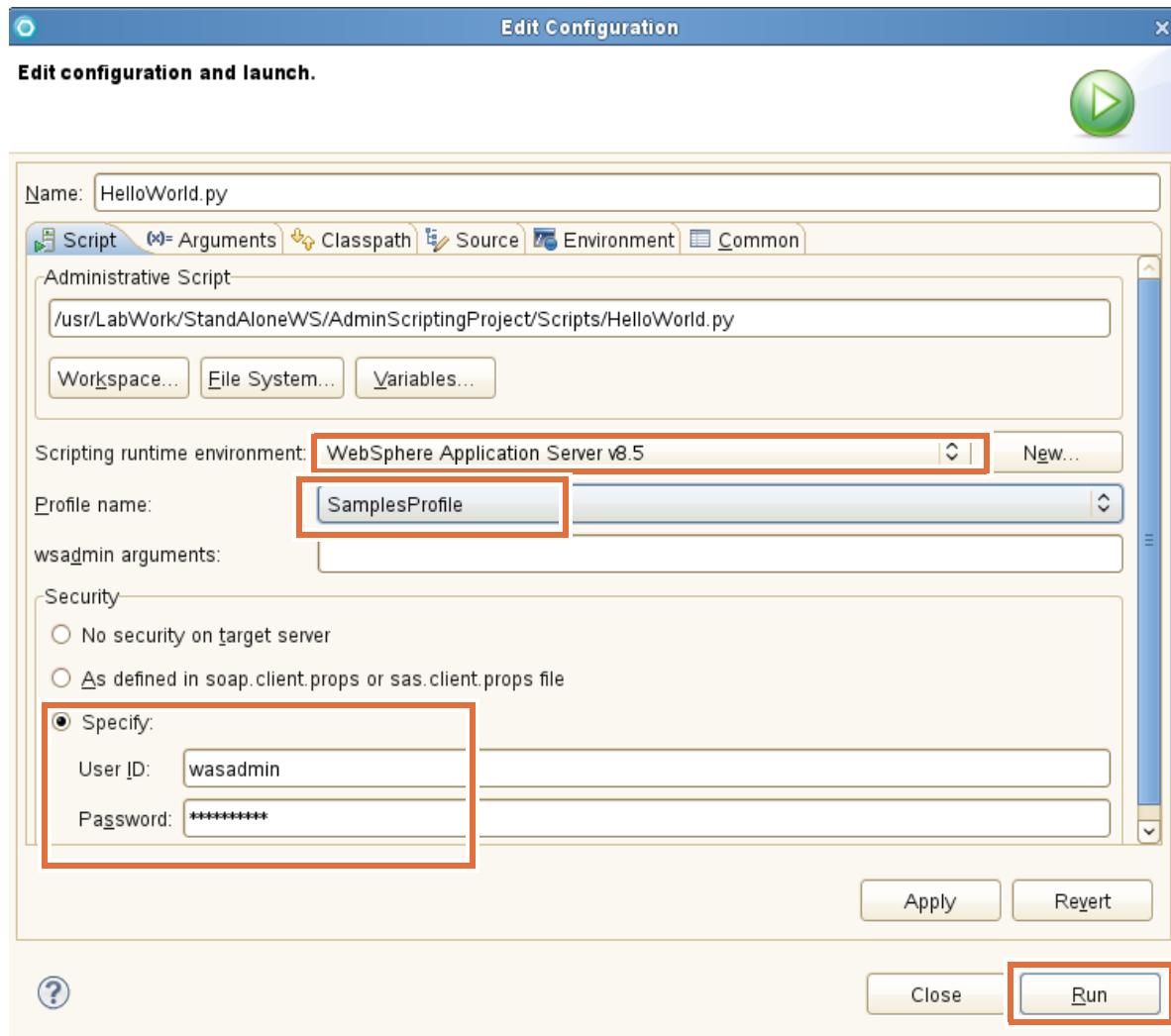
The Jython editor is not integrated with a compiler. As a result, the Jython editor does not perform syntax verification on your scripts.

- \_\_\_ 3. Run the HelloWorld.py script on the server in SamplesProfile.
- \_\_\_ a. Right-click anywhere in the Jython editor view and select **Run As > Administrative Script**

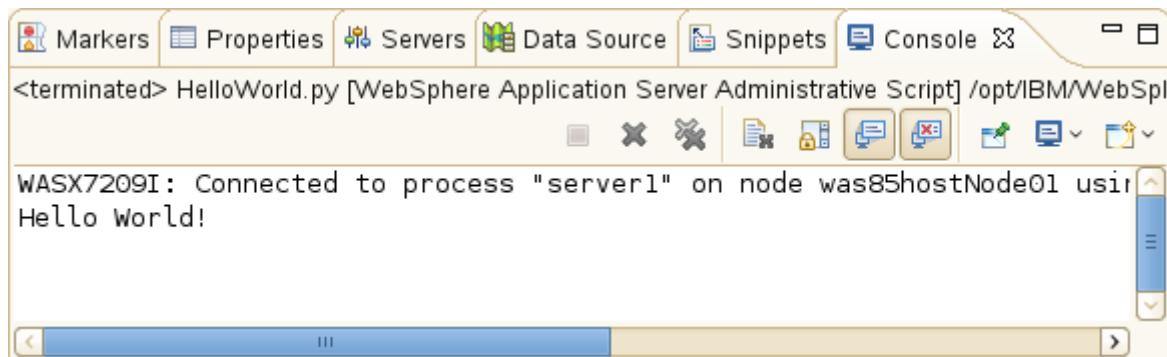


- \_\_\_ b. Since you are running the script for the first time, the **Edit configuration and launch** and window opens so that you can specify the run configuration parameters. In the dialog:
- Select **WebSphere Application Server v8.5** in the list for Scripting runtime environment.
  - Select **SamplesProfile** in the list for Profile name.

- In the Security section, select **Specify** and type `wasadmin` for the **User ID** field and `web1sphere` for the **Password** field.



- c. Click **Run** to run the script. After a few moments, the "Hello World!" string is displayed in the **Console** view.





**Note**

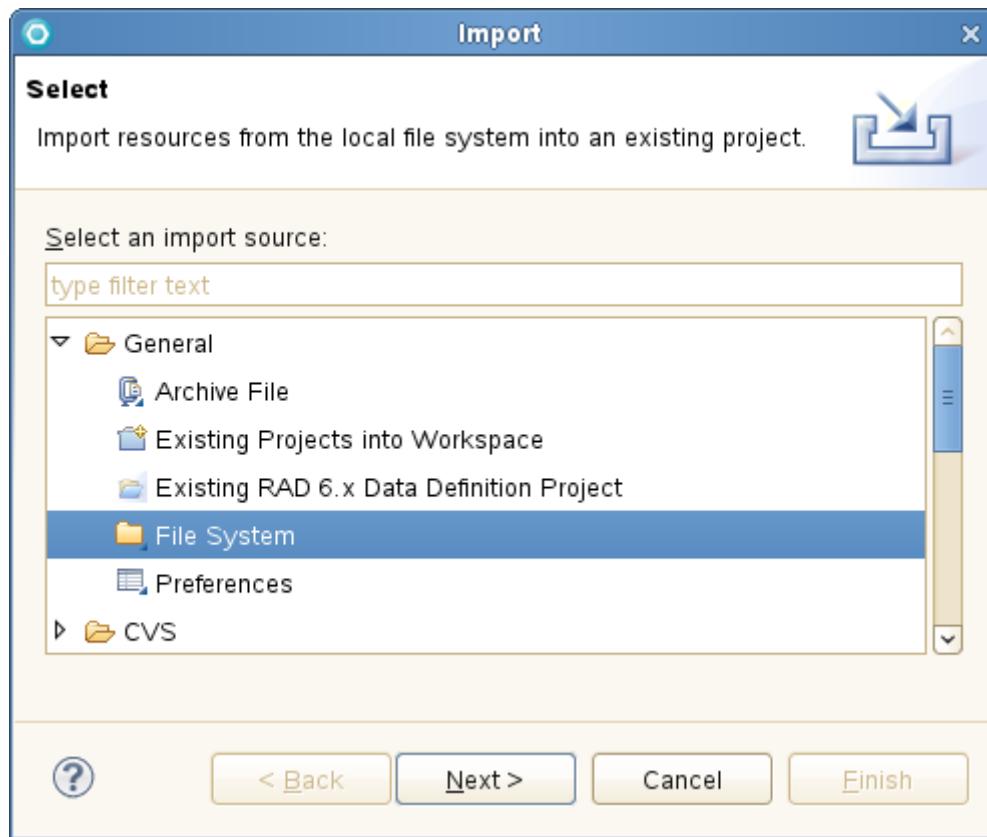
When the wsadmin launcher starts for the first time, the required Jython libraries are loaded and cached. The loaded libraries are displayed in the Console view. The screen capture does not show the libraries that are being loaded.

You successfully edited and ran a simple Jython script in IADT.

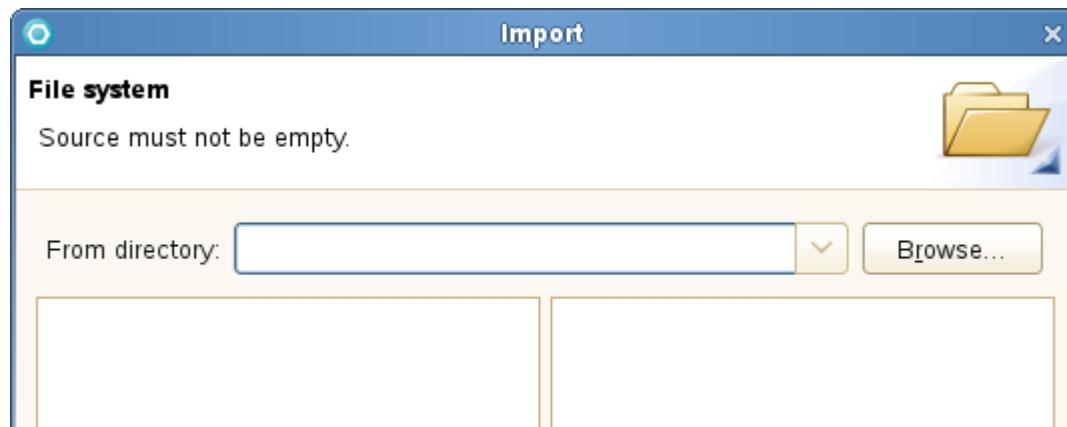
## Section 3: Exploring Jython language constructs

In this section, you explore basic Jython language constructs by importing, examining, and running several sample scripts that are provided for you. You are also introduced to the scriptExecutor utility script and begin to use it to run Jython scripts.

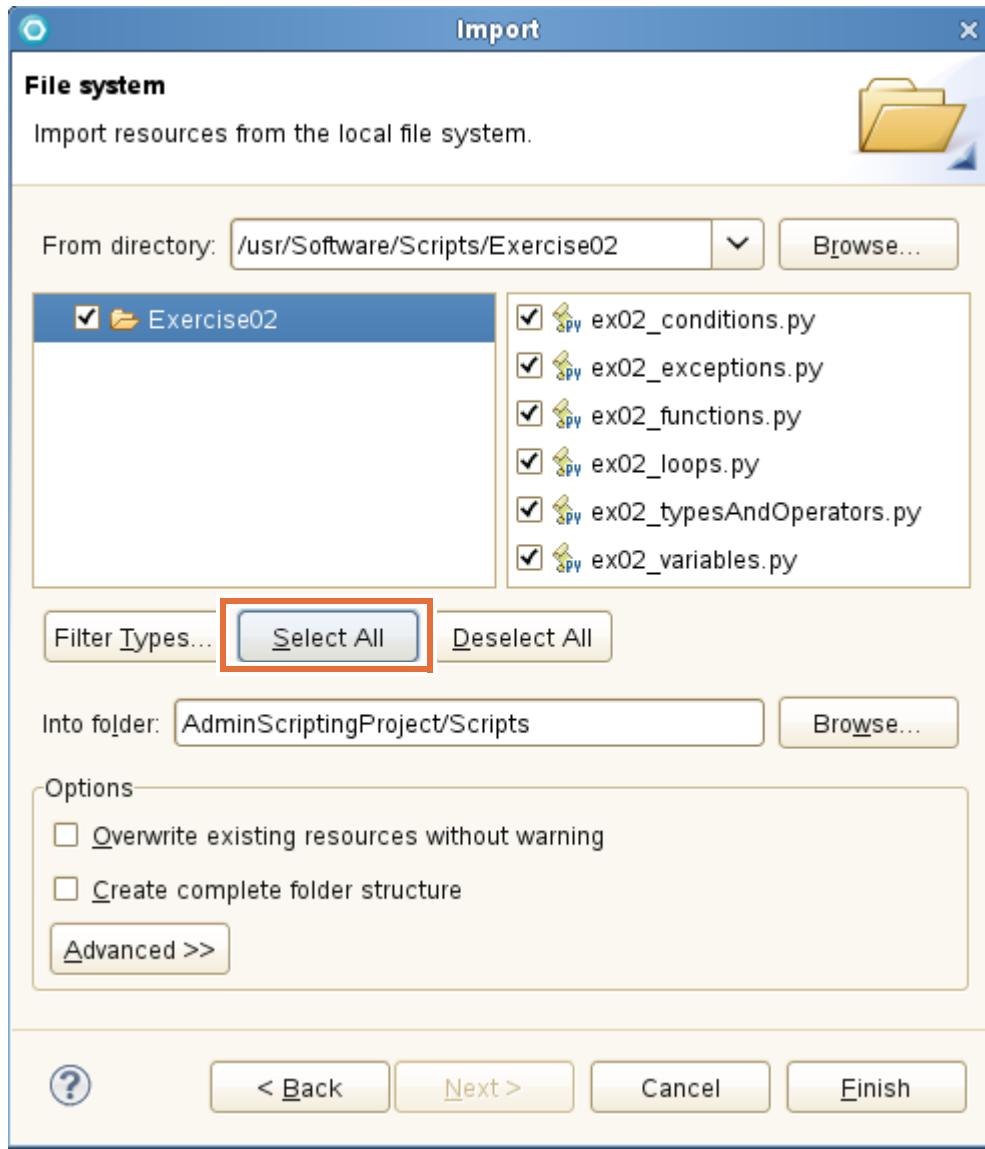
- \_\_\_ 1. Start by importing the provided sample Jython scripts.
  - \_\_\_ a. In the Enterprise Explorer view, right-click **Scripts** and select **Import**.
  - \_\_\_ b. In the **Select** dialog, expand **General** and select **File System**.



- \_\_\_ c. Click **Next**.
- \_\_\_ d. In the **File system** dialog, click **Browse** beside the "From directory" field.



- \_\_\_ e. In the **Import from directory** dialog, go to **/usr/Software/Scripts**, select **Exercise02**, and click **OK**.
- \_\_\_ f. In the **File system** dialog, select **Exercise02** and click **Select All**.



- \_\_\_ g. Click **Finish**.
- \_\_\_ h. In the Project Explorer view, expand **AdminScriptingProject > Scripts**. The following new script files now appear in the folder:

-**ex02\_conditions.py**  
-**ex02\_exceptions.py**  
-**ex02\_functions.py**  
-**ex02\_loops.py**  
-**ex02\_typesAndOperators.py**

**-ex02\_variables.py**

These script files contain code that illustrates the syntax and usage of various Jython basic language constructs.

2. Explore the **ex02\_loops.py** sample Jython script. In the Enterprise Explorer view, double-click **ex02\_loops.py** to open it in an editor view.

```
#for loop Syntax
# for elementInList in ListOfElements
alphabetList = ["a", "b", "c", "d", "e", "f", "g", "h",
alphabetString = ""

for letterElement in alphabetList:
    alphabetString = alphabetString + letterElement

print alphabetString

#While Loop Syntax
# while test1:
#     statements1
# else:
#     statements2

a = 0; b = 10
while a < b:
    print a
    a = a + 1
else:
    print "The loop is now complete"
```

This sample script includes two loop constructs. The first is a **for** loop that cycles through a list of alphabetic characters and concatenates the individual characters into a single string.

The second is a **while** loop that runs while the variable '**a**' has a value less than the variable '**b**'. When the loop ends, the final **else** clause prints a notification that the loop completed.

**Note**

The Jython editor provides color coding for keywords, wsadmin objects, strings, comments, and default text. The default color values are:

- Keywords: **magenta**
- wsadmin objects: **orange**
- Strings: **blue**
- Comments: **green**

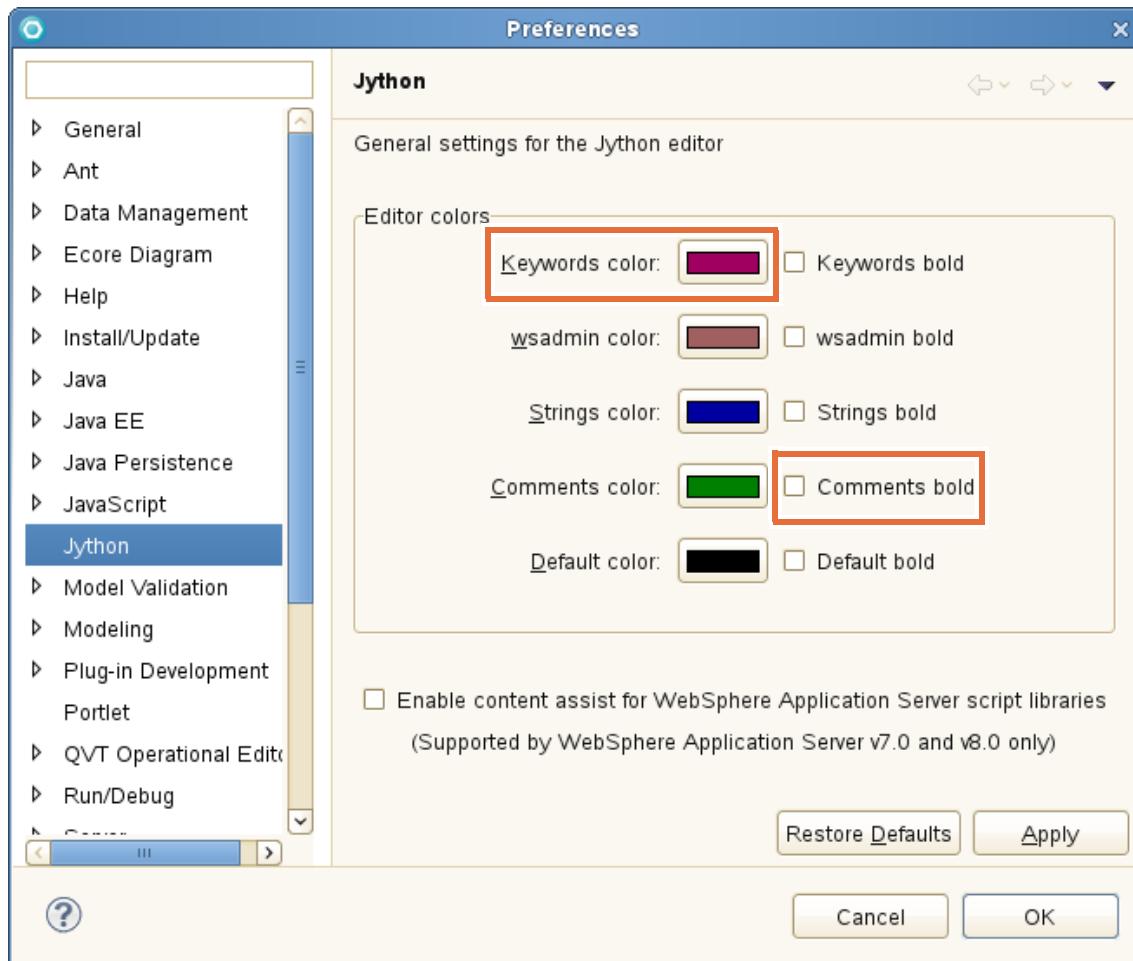
- Default: **black**



## Information

These color preferences can be changed on the Preferences window as follows:

- From the menu bar, select **Window > Preferences**.
- In the right pane, select **Jython**.
- Change the color of an item by clicking the color selector button beside its description.

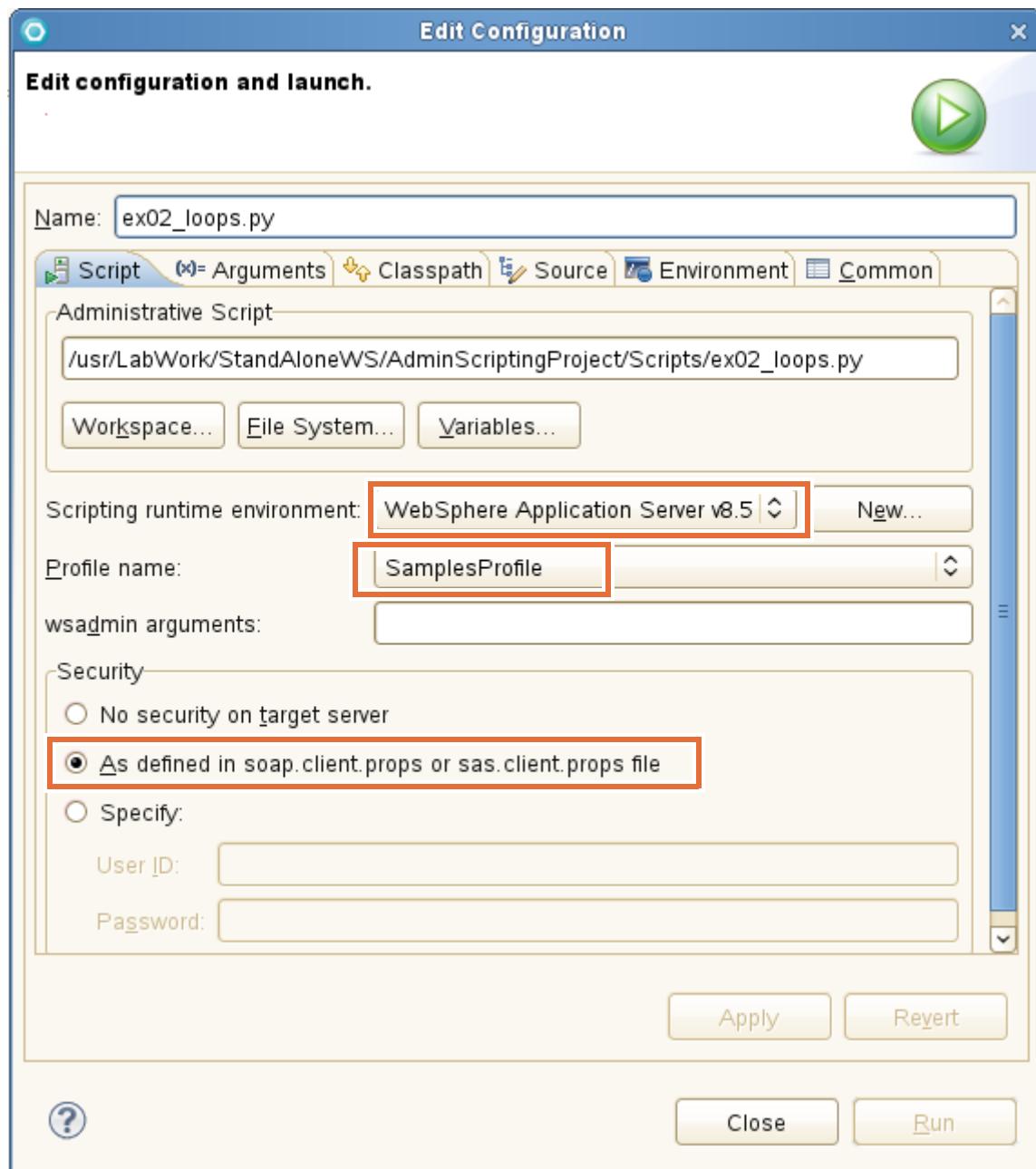


- To apply bold formatting, select the corresponding check box.
- Click **OK**.

3. Run the `ex02_loops.py` script.

- a. Right-click in the editor window and select **Run As > Administrative Script**.

- \_\_\_ b. When new scripts are run for the first time, the **Edit configuration and launch** window opens and requires you to specify the runtime environment. Use the same runtime environment that was used for the HelloWorld.py script (the server in SamplesProfile).
- In the Scripting runtime environment list, select **WebSphere Application Server v8.5**
  - In the **Profile name** list, select **SamplesProfile**
  - In the **Security** section, select **As defined in soap.client.props or sas.client.props file**.



**Note**

The **As defined in soap.client.props or sas.client.props file** uses the user name and password properties that are defined in the `soap.client.props` file from the previous lab. If you did not update the `soap.client.props` file, then select **Specify** and enter `wasadmin` and `websphere` for the **User ID** and **Password** fields.

- \_\_\_ c. Click Run.

```
abcdefghijklmnopqrstuvwxyz
0
1
2
3
4
5
6
7
8
9
The loop is now complete
```

The resulting execution of the `ex02_loops.py` script is displayed in the Console view.

- \_\_\_ 4. Now, modify the `ex02_loops.py` script to see what happens if you “accidentally” indent one of the lines in the `for` loop.
- \_\_\_ a. In the `ex02_loops.py` editor view, indent the following line:

```
print alphabetString
```

```
# ex03_loops.py

#for loop Syntax
# for elementInList in ListOfElements
alphabetList = ["a", "b", "c", "d", "e", "f", "g", "h",
alphabetString = ""

for letterElement in alphabetList:
    alphabetString = alphabetString + letterElement

→ print alphabetString
```

- \_\_\_ b. Type **Ctrl+S** to save the change.

- \_\_\_ c. Run the `ex02_loops.py` script again. Because the tool remembers the last run configuration that was run (and it did for this script), you can click **Run** on the button bar to run it.

```

WASX7209I: Connected to process "server1" on node was85hostNode01 using
a
ab
abc
abcd
abcde
abcdef
abcdefg
abcdefgh
abcdefghi
abcdefhij
abcdefhijk
abcdefhijkl
abcdefhijklm
abcdefhijklmn
abcdefhijklmno

```

Notice that the Console view now shows a different output for the `for` loop. This example illustrates the importance of correct indentation in Jython.

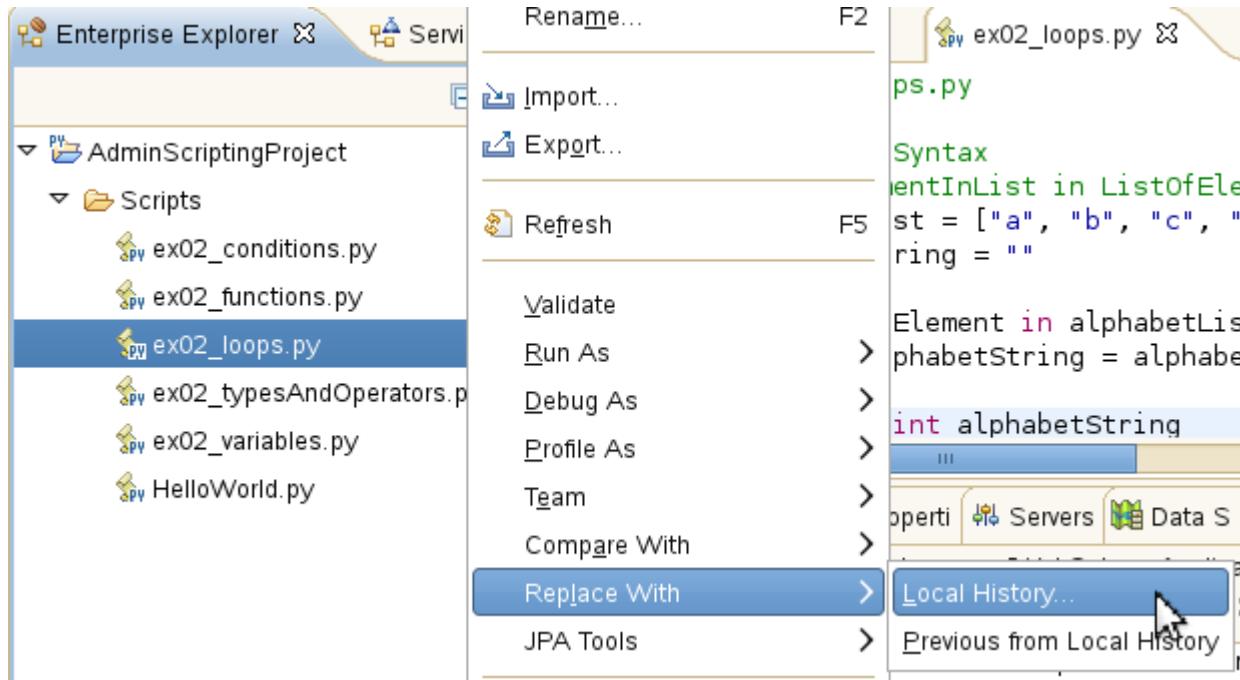


### Information

In Jython, the indentation level of your statements is significant. The exact amount of indentation does not matter, only the indentation of nested blocks relative to each other.

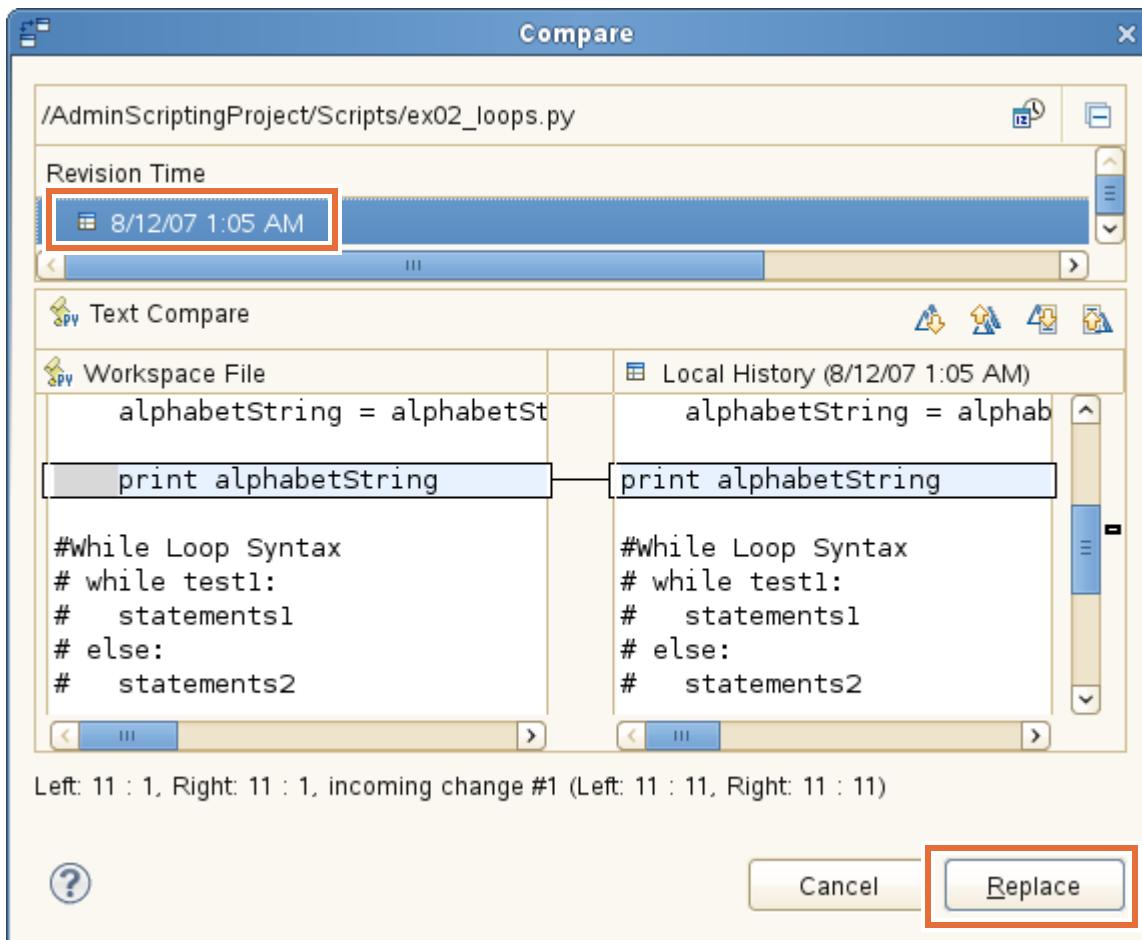
The indentation level is ignored if you use explicit or implicit continuation lines. For example, you can split a string across multiple lines by using the '\ ' (backslash) character (explicit). Or you can press the Return key to start a new line (implicit).

5. Restore the ex02\_loops.py script to its correct version. You can use the “Local History” feature of IADT to quickly restore the script.
- a. In the Enterprise Explorer view, right-click **ex02\_loops.py** and select **Replace with > Local History**. The Replace from Local History window opens.



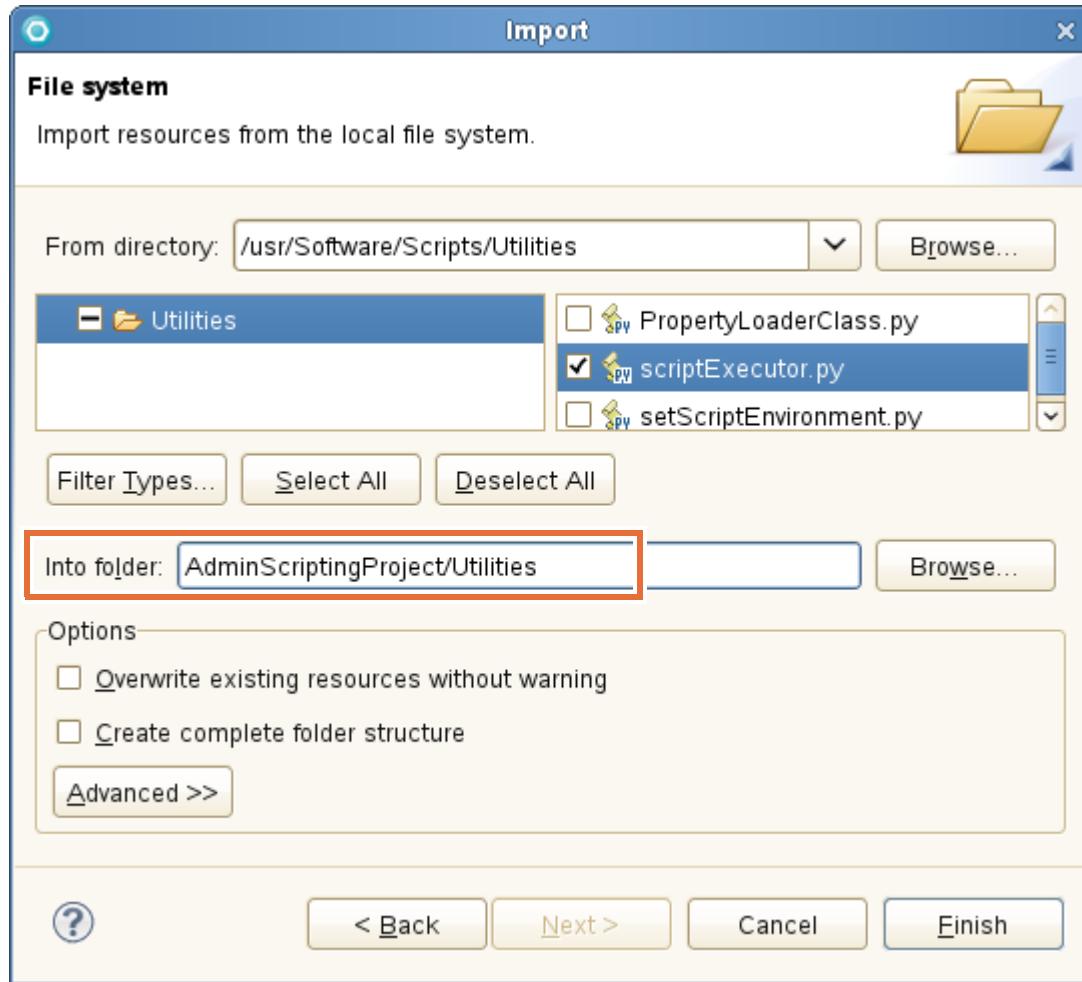
The top pane lists the various timestamps for each time you saved the file. Each represents a version of the file that you can revert to. When you select a timestamp, the bottom panes show a comparison of the current file contents with the contents of the selected snapshot. In your case, you have one snapshot since you only saved the file once.

- \_\_\_ b. Double-click the timestamp..



- \_\_\_ c. Click **Replace** to revert to the selected snapshot. The editor view confirms that your script is restored to its original version.
- \_\_\_ 6. So far, you should notice that every time you ran a script, you had to wait a moment before it actually ran and produced its output in the Console view. This behavior is because each time a script is run in the IADT, a new wsadmin session is created. This leads to significant delays, especially when you must run a script multiple times while testing it. To remedy this situation, a Jython utility script was developed for this course to accelerate the execution of Jython scripts or individual commands inside the IADT called `scriptExecutor`. The `scriptExecutor.py` utility script avoids wsadmin startup wait times by establishing a single wsadmin runtime session that is repeatedly reused to run other Jython scripts or commands. Import the `scriptExecutor` utility to use it to run the other sample scripts.
- \_\_\_ a. In the Enterprise Explorer view, right-click **AdminScriptingProject** and select **Import**.
- \_\_\_ b. In the **Import select** dialog, expand **General** and select **File system**.
- \_\_\_ c. Click **Next**.
- \_\_\_ d. In the **File system** dialog, click **Browse** next to the From directory field.

- \_\_\_ e. In the **Import from directory** dialog, go to **/usr/Software/Scripts**, select **Utilities**, and click **OK**.
- \_\_\_ f. In the **File system** dialog:
- Select the **scriptExecutor.py** check box.
  - Type **/Utilities** at the end of **AdminScriptingProject** in the **Into folder** field. This action causes the file to be imported in a folder named Utilities under the project.



- Click **Finish**.
- \_\_\_ g. In the Enterprise Explorer view, expand **AdminScriptingProject > Utilities**. The **scriptExecutor.py** is displayed in the folder.
- \_\_\_ h. Take a moment to glance over the code for the scriptExecutor script.



## Note

You are not expected to immediately understand all of the code in the **scriptExecutor** script. The purpose of this step is to give you a brief overview of its inner workings. You do not have to fully understand the implementation of the **scriptExecutor** to use it.

- In the Enterprise Explorer view, double-click **scriptExecutor.py** to open it in an editor view. A brief explanation of its content is provided.

The `scriptExecutor.py` utility defines an `executeScript` function that accepts two parameters that are retrieved from command-line arguments. The first specifies the working directory for the scripts and is required. The second indicates a default Jython script to run, when none is specified, and is optional.

An infinite while loop runs and prompts the user to input the name of a Jython script file. This file name is passed to an `execfile()` method, which runs the script.

Alternatively, a Jython command can be passed to the `scriptExecutor.py` utility, which runs it using an `exec()` method.

If there are any errors in the script or command input, the `scriptExecutor.py` prints an error message, which includes the execution stack trace, and prompts the user to enter another script file or command to run.

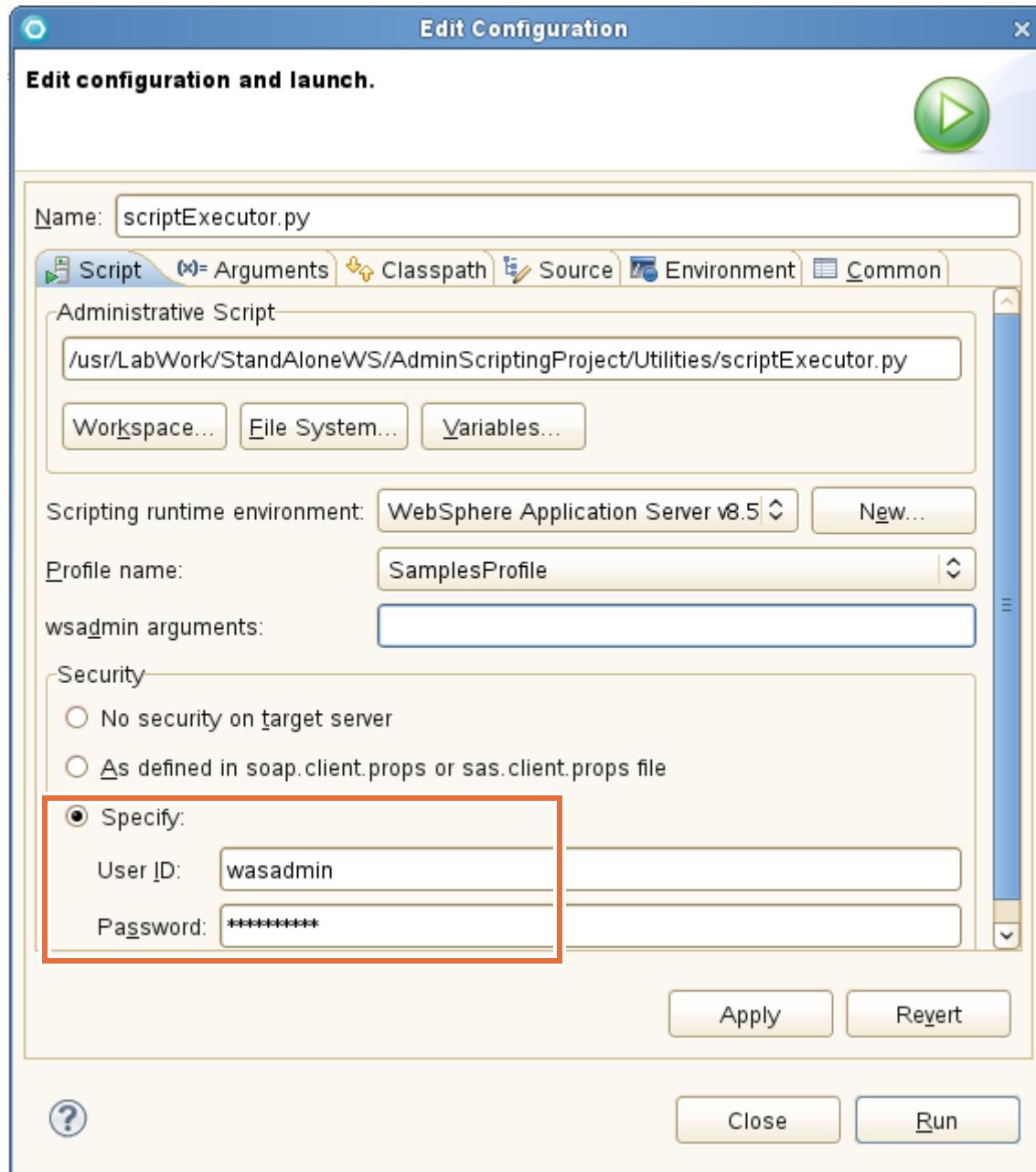
To exit the `scriptExecutor.py` script, type `quit` and press the **Enter** key.

- Close the editor view by clicking the 'x' next to the **scriptExecutor.py** tab label.

```
from java.lang import Throwable
from java.util import Properties
from java.io import FileInputStream
```

- \_\_\_ 7. Run the **scriptExecutor** utility script.
  - \_\_\_ a. In the Enterprise Explorer view, expand **AdminScriptingProject > Utilities** if necessary. Right-click **scriptExecutor.py** and select **Run As > Administrative Script**.
  - \_\_\_ b. Since you are running the script for the first time, the **Edit configuration and launch** window opens so that you can specify the run configuration parameters. In the dialog:
    - Select **WebSphere Application Server v8.5** in the list for Scripting runtime environment.
    - Select **SamplesProfile** in the list for Profile name.

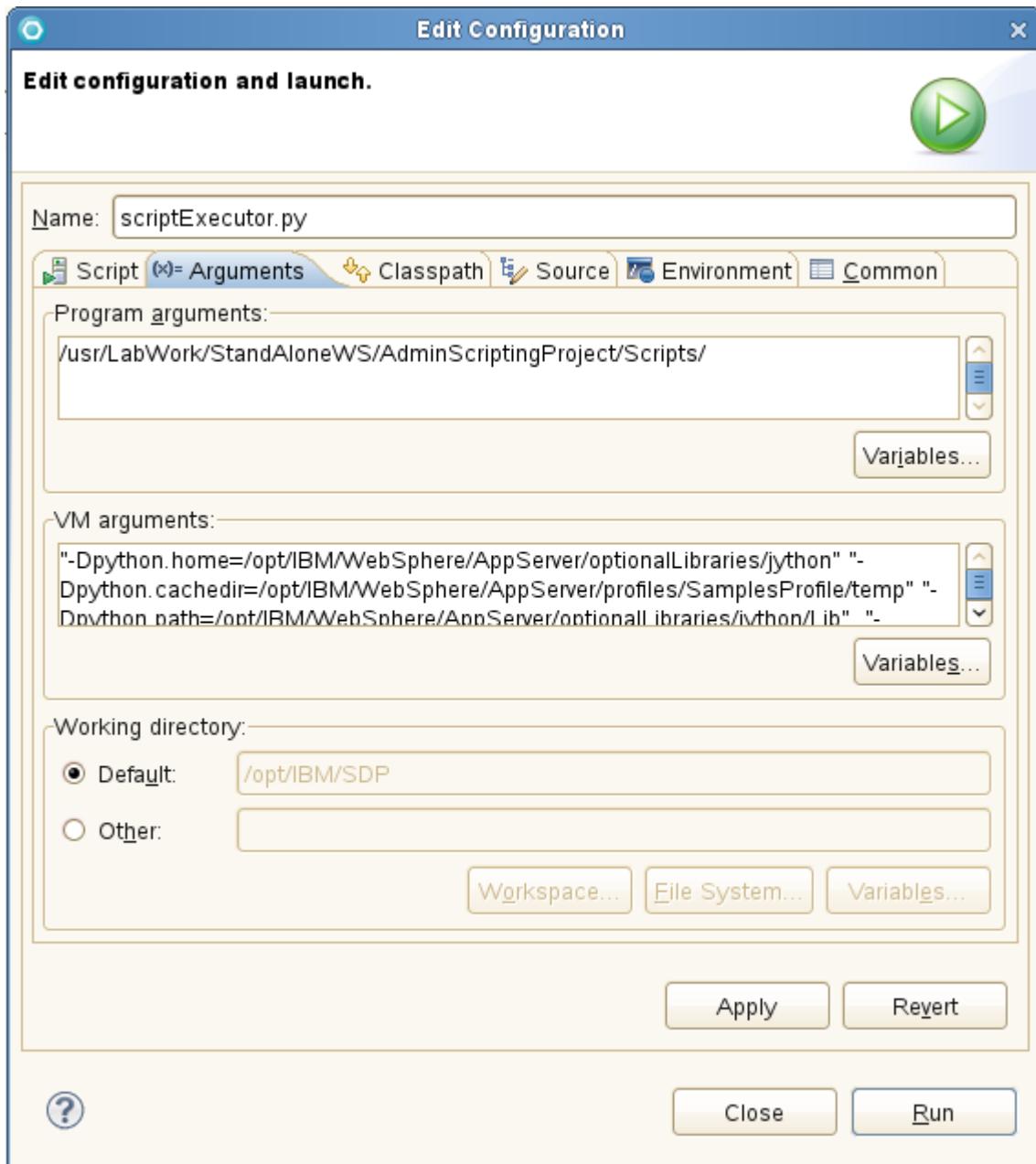
- In the Security section, select **Specify** and type `wasadmin` for the **User ID** field and `websphere` for the **Password** field.



- Select the **(x)=Arguments** tab to specify an argument to pass to the `scriptExecutor` script.
- c. On the **(x)=Arguments** tab, type `/usr/LabWork/StandaloneWS/AdminScriptingProject/Scripts/` in the Program arguments field. This argument specifies the location where the `scriptExecutor` looks for scripts to run.

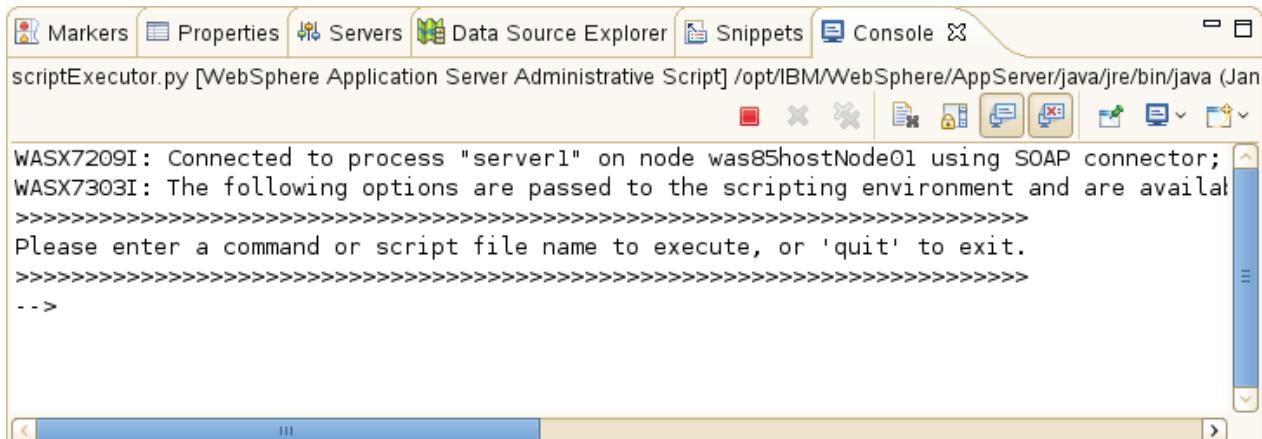
**Important**

Make sure to type forward slashes and include the last slash.



- \_\_\_ d. Click **Run** to run the script. After a few moments, the **Console** view displays a message that confirms the establishment of a wsadmin connection with the

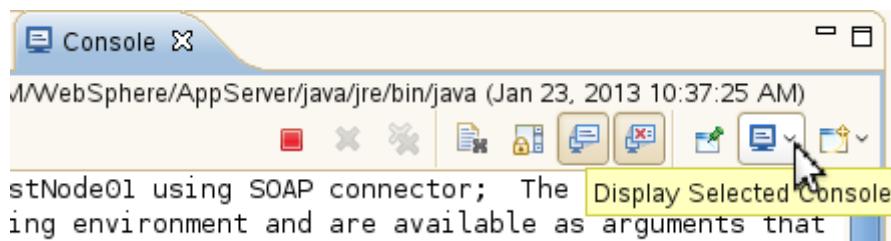
**server1** server. An input prompt is also displayed for running an administrative command or a script file. You are now ready to run your scripts.



## Information

## Switching console views

You have two Console views opened: one for the **scriptExecutor** script and the other of the *server1* standard output. To switch between Console views, click the drop-down arrow of the **Display Selected Console** button and select the console output to view.



- \_\_ e. Test the scriptExecutor by running the `ex02_conditions.py` script.

  - First, open `ex02_conditions.py` in an editor view and review its code. It contains a simple Jython `if-elif-else` statement that prints different messages that are based on the `adminChoice` condition.
  - In the scriptExecutor Console view, type `ex02_conditions.py` and press the Enter key.



## Information

When you gain focus on the scriptExecutor Console view, the cursor is not immediately positioned on the input prompt line. However, you can start typing a command and the cursor is automatically repositioned on the input prompt line.

```

Markers Properties Servers Data Source Expl Snippets Console
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/
Please enter a command or script file name to execute, or 'quit' to exit.
-->ex02_conditions.py
wsadmin is an excellent scripting utility.

Please enter a command or script file name to execute, or 'quit' to exit.
Or press the 'Enter' key to execute the ex02_conditions.py script.
-->

```

The `ex02_conditions.py` script runs and the resulting output is displayed. The `scriptExecutor.py` script continues to run and prompts for another script to run.

- \_\_\_ 8. Review and run the remaining sample scripts. For each of the following scripts:
  - `ex02_functions.py`
  - `ex02_typesAndOperators.py`
  - `ex02_variables.py`
- \_\_\_ a. Open the script in the Jython editor and review its code content.
- \_\_\_ b. Run the script by using the `scriptExecutor` utility.
- \_\_\_ c. Examine the results in the `scriptExecutor` Console view.
- \_\_\_ 9. Leave the `scriptExecutor` running as you continue to use it in the next section.

## Section 4: Developing an administrative script

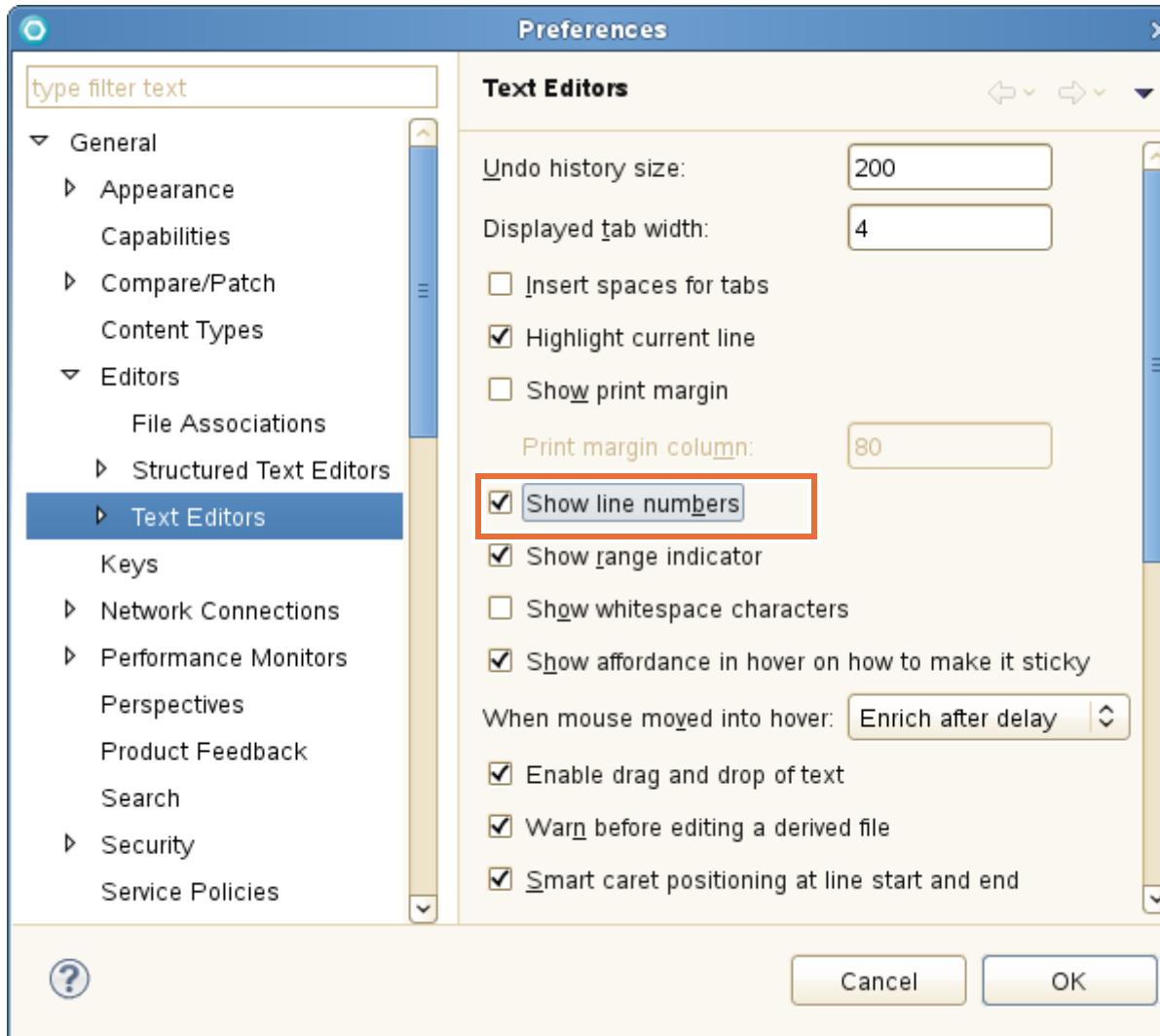
In this section, you develop a Jython script that performs the administrative functions of:

- Listing all of the installed applications on a server
- Listing the modules inside an application that is selected from the list.

In addition to allowing you to use the Jython constructs that you learned in the previous section, this section introduces the use of the content assist feature of Jython editor and the AdminApp administrative scripting object.

- \_\_\_ 1. Create another script file named `ex02_exploreWebSphereApps.py`.
  - \_\_\_ a. In the Enterprise Explorer view, right-click the **Scripts** folder and select **New > Other**. The **Select a wizard** dialog opens.
  - \_\_\_ b. In the **Select a wizard** dialog, expand **Jython** and select **Jython Script File**. Click **Next**.
  - \_\_\_ c. In the **Create a Jython script file** dialog, type `ex02_exploreWebSphereApps.py` in the File name field.
  - \_\_\_ d. Click **Finish**. The new script file is now shown in the Enterprise Explorer view in the Scripts folder, and is opened in the Jython editor view.
- \_\_\_ 2. It is useful to see line numbers in the editor, so change your preferences to show them.
  - \_\_\_ a. In the main menu bar, select **Window > Preferences**. The Preferences window opens.
  - \_\_\_ b. In the left pane, expand **General > Editors**, and select **Text Editors**.

- \_\_ c. In the right pane, select the **Show line numbers** check box.



- \_\_ d. Click **OK**. Line numbers are now shown in the left column in the Jython editor view.  
 \_\_ 3. Now, enter the code for the `exploreWebSphereApps.py` script. The code is provided for you.



### Information

The completed version of the `ex02_exploreWebSphereApps.py` script is available in the `/usr/Solutions/Exercise02` directory. You can copy and paste it into the Jython editor.

- \_\_ a. Start by removing the three comment lines that include the `TODO: enter JYTHON` code and save comment (lines 1-3).

- \_\_ b. Type the code that is displayed following the **Note** about using content assist and content tip in the Jython editor:



## Using Content Assist and Content Tip

In the Jython editor, use content assist and content tip to speed up the creation of WebSphere administrative (wsadmin) objects, methods, parameters, and Jython syntax.

Content assist helps you finish a line of code in the Jython editor. The placement of the cursor in your script file provides the context for the content assist to offer a list of suggested completion. The content assist for administrative objects, methods, and parameters displays information from the available Javadoc, while the content assist for Jython displays only the keyword syntax.

## To use content assist:

- 1. Place your cursor in a partially entered string of code and type **Ctrl+Space** or select **Edit > Content Assist** from the main menu bar. If the Jython editor finds valid candidates for this position, a list of possible completions is shown in a floating window. You can type further to narrow the list.
  - 2. Use the arrow keys or the mouse pointer to go through the possible completions.
  - 3. Select an item in the list to complete the code fragment by selecting it and pressing Enter or double-clicking it.

The screenshot shows a Python code editor with two tabs: 'ex02\_variables.py' and '\*ex02\_exploreWebSphe'. The code in 'ex02\_variables.py' is as follows:

```
1 #ex02_exploreWebSphereApps.py
2 import os
3 def isDigit(value):
4     if str(value).isdigit() != 1:
5         return "false"
6     return "true"
7
8 appList=AdminApp.list(
9
10
11 list().split(lineSepar
12
13 while 1:
14     print 'Install
15     for i in range
16         print
17
18         print "\n"
19
20         inputValue= ra
21         if isDigit(inp
22             try:
23
24             print appList[int(inputValue)], "Modules"
25         except:
26             print "Error: Invalid input"
```

A code completion dropdown is open over the line 'appList=AdminApp.list('. It lists several suggestions:

- AdminApp.list ()
- AdminApp.list (target scope)
- AdminApp.listModules (application name, options)
- AdminApp.listModules (application name)

The suggestion 'AdminApp.list ()' is highlighted in blue. To its right, a tooltip provides the documentation: 'Lists all installed applications'.

Content tip allows you to see a list of parameter hints in a method argument. If the cursor is positioned at the parameter specification for a method reference, this action shows a floating window with parameter type information.

To use content tip, place your cursor in a parameter specification for a method reference and start the content tip by typing **Ctrl+Shift+Space** or selecting **Edit > Content Tip** from the main menu bar. If the Jython editor finds valid candidates for this position, a list of parameter hints is shown in a floating window.

The screenshot shows a Jython script editor window with two tabs: "ex02\_variables.py" and "\*ex02\_exploreWebSphe". The code in "ex02\_variables.py" is as follows:

```
1 #ex02_exploreWebSphereApps.py
2 import os
3 def isDigit(value):
4     if str(value).isdigit() != 1:
5         return "false"
6     return "true"
7
8 appList=AdminApp.list().split(lineSeparator)
9
10 while 1:
11     print 'Install'
12     for i in range
13         print
14
15     print "\n"
16
17     inputValue= ra
18     if isDigit(inp
19         try:
20
21             print AdminApp.listModules (appList[int(inp
22
23
```

A content tip floating window is displayed over the code, centered on the line "print AdminApp.listModules (appList[int(inp". The window lists several method candidates for "AdminApp.list":

- AdminApp.list ()
- AdminApp.list (target scope)
- AdminApp.listModules (application name, options)
- AdminApp.listModules (application name)

The floating window has a semi-transparent background and a scroll bar on the right side. A small "x" button is visible in the top right corner of the window.

```
import os

def isDigit(value):
    if str(value).isdigit() != 1:
        return "false"
    return "true"

appList=AdminApp.list().split(os.linesep)

while 1:
    print 'Installed applications:'
    for i in range (len(appList)):
        print i, ', ', appList[i]

    print "\n"

    inputValue= raw_input( "Choose an application from above to
                           list its modules: ")

    if isDigit(inputValue) == "true" :
        try:
            print "\n"
            print appList[int(inputValue)], "Modules"
            print "-----"
            print AdminApp.listModules (appList[int(inputValue)])
            break

        except:
            print "An error occurred while listing modules for app ", inputValue
        #end try

    else:
        print "A value between 0 and ", len(appList)-1, " must be entered."
    #end if
    import time
    time.sleep(5)
#end while
```

**Important**

The indentation in a Jython script is important. Pay particular attention to function definitions, while/for loops, conditional statements, and exception blocks. Misplaced indentation leads to incorrect execution.

```

1 import os
2
3 def isDigit(value):
4     if str(value).isdigit() != 1:
5         return "false"
6     return "true"
7
8 appList=AdminApp.list().split(os.linesep)
9
10 while 1:
11     print 'Installed applications:'
12     for i in range (len(appList)):
13         print i, ', ', appList[i]
14
15     print "\n"
16
17 inputValue= raw_input( "Choose an application from above to list their modules: ")
18
19 if isDigit(inputValue) == "true" :
20     try:
21         print "\n"
22         print appList[int(inputValue)], "Modules"
23         print "-----"
24         print AdminApp.listModules (appList[int(inputValue)])
25         break
26
27     except:
28         print "An error occurred while listing modules for app ", inputValue
29     #end try
30
31 else:
32     print "A value between 0 and ", len(appList), " must be entered."
33 #end if
34
35 import time
36 time.sleep(5) sleep timer
37 #end while

```

**Method definition**

**for loop**

**User input**

**exception block**

The following is an explanation of the code:

- **Line 1** import the operating system class named **os**.
- **Line 3** defines a function named **isDigit()** that accepts a string argument. The method, in turn, uses the built-in string method **isdigit()** to verify whether the passed string argument is an integer value. If the value is an integer, a value of “true” is returned, otherwise “false” is returned.
- **Line 8** uses the **AdminApp.list()** method to return a list of the installed applications. The returned value is actually a string that contains application

names that are separated by “\r\n” characters. To convert it to a Jython list, the `split(os.linesep)` function is used. It removes the “\r\n” separator characters and converts the string into a Jython list, which is stored in the `appList` variable. The `os.linesep` variable is defined in the Jython `os` module and represents the operating system characters for a line separator (“\r\n” on Windows).

- **Line 10** starts an infinite `while` loop that runs until the `break` statement is called when the script successfully completes. The `break` statement is called when the user chooses an application whose modules are displayed.
- **Line 12** uses the `for` loop to cycle through the list of installed applications and print a listing of the application name along with numeric selection value assigned to each name.
- **Line 17** calls the `raw_input(prompt)` method to display the prompt and retrieve user input. This method returns a string that contains the user input. An alternative method, `input(prompt)`, is available to capture user input and immediately run it using the Jython interpreter.
- **Line 19** is an `if` statement, which calls the `isdigit()` function to verify that the value entered by the user is indeed a numeric value.
- **Line 20** defines the `try/except` exception block that is used to catch any unexpected errors. For instance, if the user enters a numeric value greater than the number of installed applications, an exception is thrown. The `except` block catches it, prints an error message, and allows execution to continue.
- **Lines 21-24** run a set of print statements to display the list of modules inside the specified application.
- **Line 25** calls the `break` statement to end the execution of the `while` loop.

- \_\_\_ c. When finished typing all of the code, type **Ctrl+S** to save the changes.
- \_\_\_ 4. Run the **ex02\_exploreWebSphereApps.py** script with the scriptExecutor.

- \_\_\_ a. In the scriptExecutor Console view, enter:

```
ex02_exploreWebSphereApps.py
```



### Note

If you have errors in your script, the scriptExecutor Console view displays corresponding error messages. Review them and make any necessary corrections to the script. Save the script and run it again. Repeat these steps until the script runs correctly.

- \_\_ b. A list of installed applications is displayed. When prompted, enter a numeric value that is associated with one of the installed applications and press Enter.

The list of modules that are contained in the selected application is returned.

- \_\_\_ c. Press Enter to run the ex02\_exploreWebSphereApps.py script again. This time, enter an out-of-bound value such as 7 in the selection prompt. Did the code handle the exception gracefully?
  - \_\_\_ d. Continue to test different code paths in the script. For example, you might want to enter a non-numeric value at the prompt to see whether it is handled properly.
  - \_\_\_ e. When you are satisfied that the script runs correctly, stop the scriptExecutor utility by typing `quit` and pressing Enter in the scriptExecutor Console view.

You developed and tested a Jython script that performs simple administrative functions.

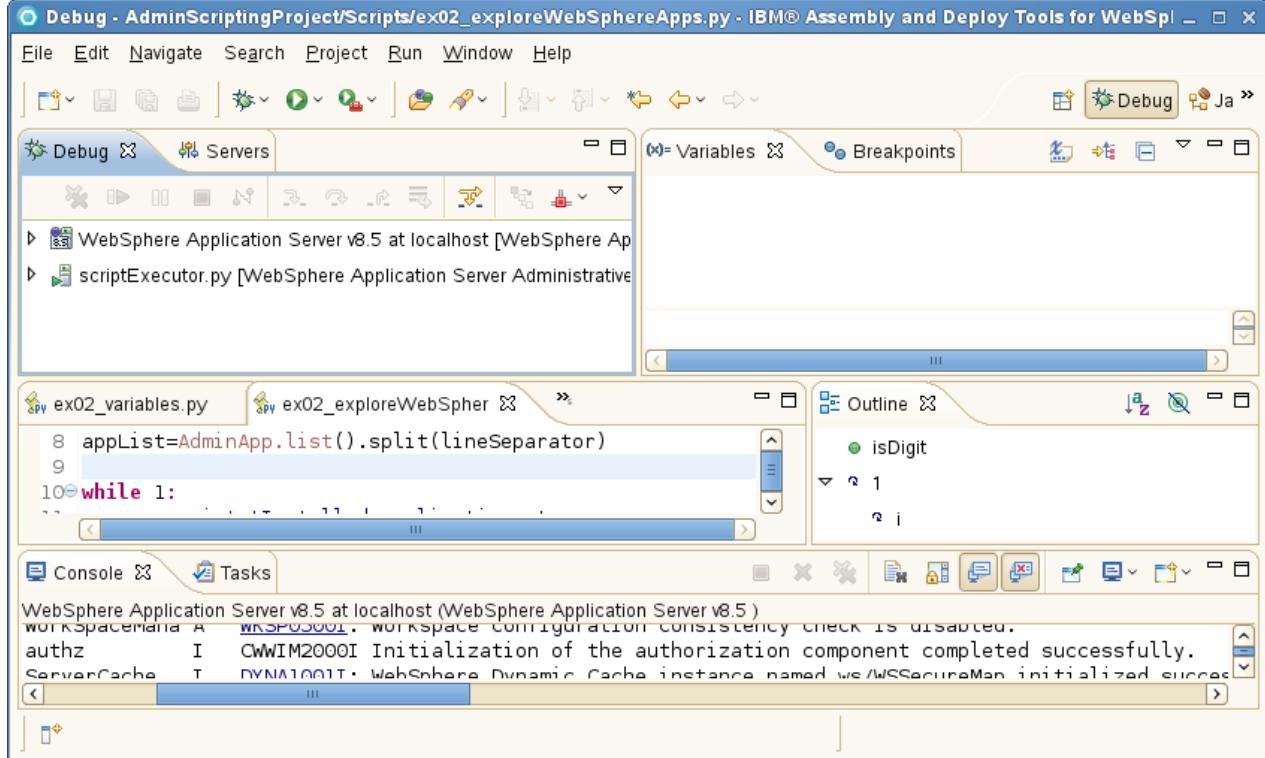
## Section 5: Debugging a Jython script

The IADT provides a debugger that enables you to detect and diagnose errors in Jython scripts. It allows you to control the execution of your script by setting breakpoints, suspending threads, stepping through the code, and examining the contents of variables.

In this section, you set breakpoints in the `ex02_exploreWebSphereApps.py` script that you developed earlier and run it in debug mode. You familiarize yourself with the debugger features as you step through the code and view variables.

- 1. Start by opening the **Debug** perspective, which contains the views that you would use while debugging.
  - a. From the main menu bar, select **Window > Open Perspective > Debug**.

The Debug perspective opens.

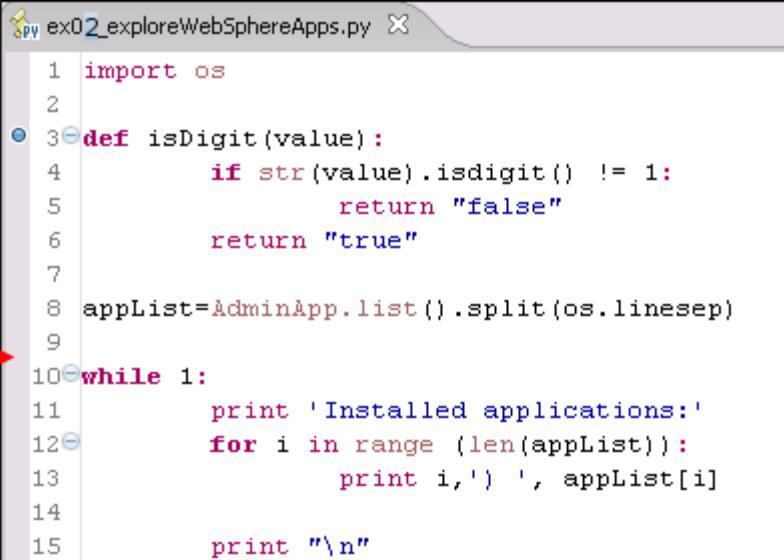


The Debug perspective is composed of several views that help you debug scripts that include:

- **Debug:** This view allows you to manage the debugging or running of a script. It displays the stack frame for the suspended threads for each target you are debugging. Each thread in your script is shown as a node in the tree. It displays the process for each target you are running.
- **Breakpoints:** This view lists all the breakpoints that are set in the workbench projects. You can double-click a breakpoint to display its location in the editor. In this view, you can also enable or disable breakpoints, delete them, or add new ones.

- **Variables:** This view displays information about the variables in the currently selected stack frame.
- **Editor:** This view is used for editing, viewing, and setting breakpoints in Jython scripts. When debugging scripts, execution pauses at the specified breakpoints in the editor.
- **Console:** This view shows the output of a process and allows you to provide keyboard input to a process. The console shows three different kinds of text, each in a different color.

- \_\_\_ 2. Now, set several breakpoints in the `ex02_exploreWebSphereApps.py` script.
- \_\_\_ a. In the `ex02_exploreWebSphereApps.py` editor view, locate the following line of code:
- ```
def isDigit(value) :
```
- \_\_\_ b. Double-click in the **marker bar** at the function definition line to set a breakpoint.



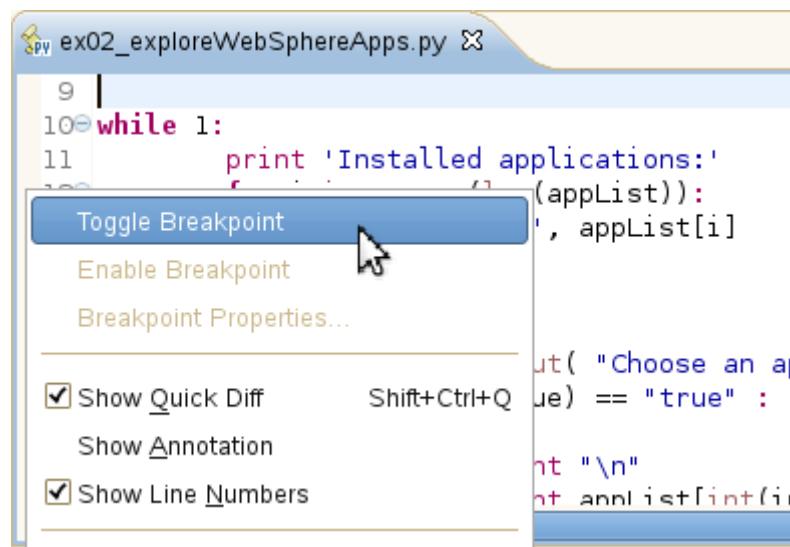
```
1 import os
2
3 def isDigit(value):
4     if str(value).isdigit() != 1:
5         return "false"
6     return "true"
7
8 appList=AdminApp.list().split(os.linesep)
9
10 while 1:
11     print 'Installed applications:'
12     for i in range (len(appList)):
13         print i,' ', appList[i]
14
15     print "\n"
```

A blue sphere appears in the marker bar to indicate that the breakpoint is set.

- \_\_\_ c. Locate the following line of code:

```
for i in range (len(appList)):
```

- \_\_\_ d. Right-click in the marker bar at the level of the line, and select **Toggle Breakpoint**. This method is another way to set breakpoints.



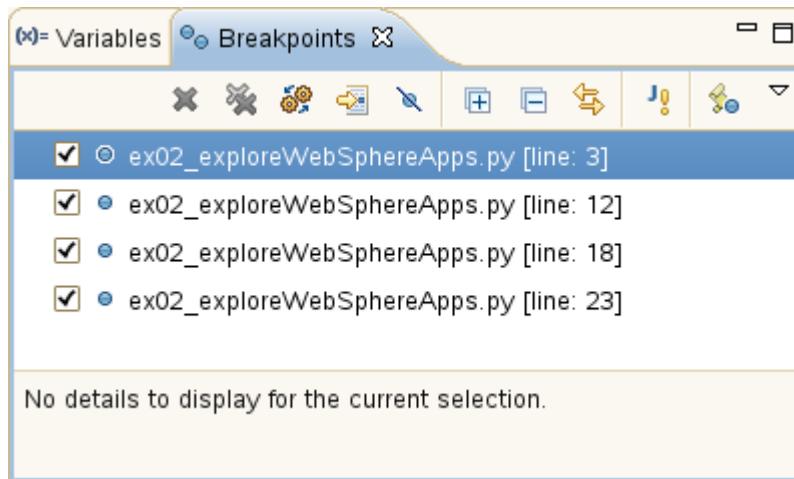
- \_\_\_ e. Using either of these techniques, locate and set a breakpoint on the following two lines:

```
if isDigit(inputValue) == "true" :
.
.
.

print AdminApp.listModules (appList[int(inputValue)])
```

The screenshot shows the same Python file 'ex02\_exploreWebSphereApps.py' in the Eclipse IDE. Two specific lines have been marked with red circles, indicating they are breakpoints: line 18 ('if isDigit(inputValue) == "true" :') and line 23 ('print AdminApp.listModules (appList[int(inputValue)])'). The code itself prints a list of application modules.

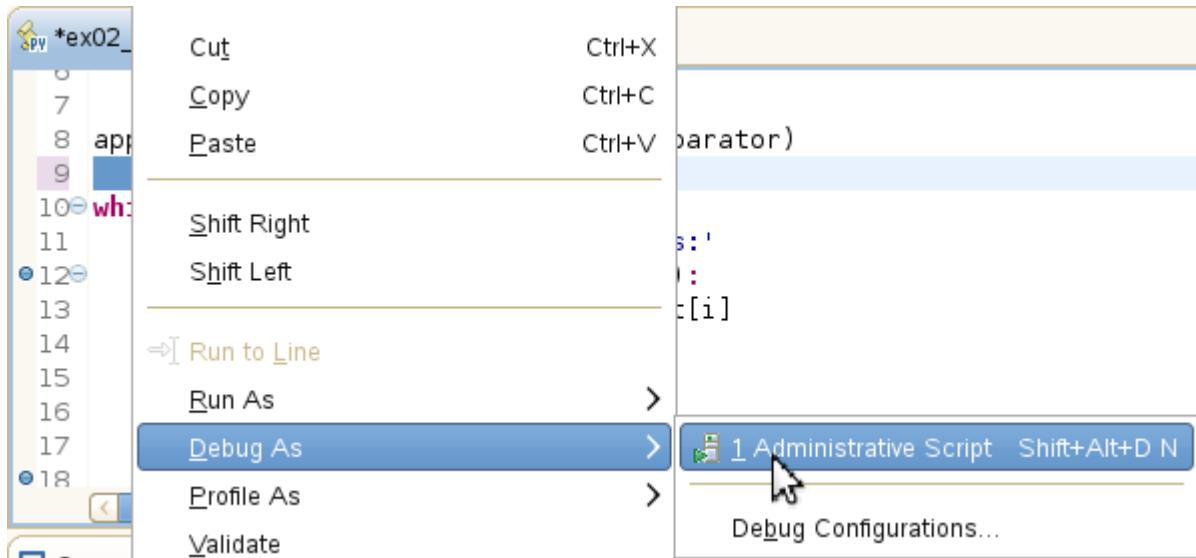
- \_\_\_ f. You can use the Breakpoints view to see the breakpoints that you set. Click the **Breakpoints** tab label to gain focus on the view. You should see the four breakpoints that you set.



### Information

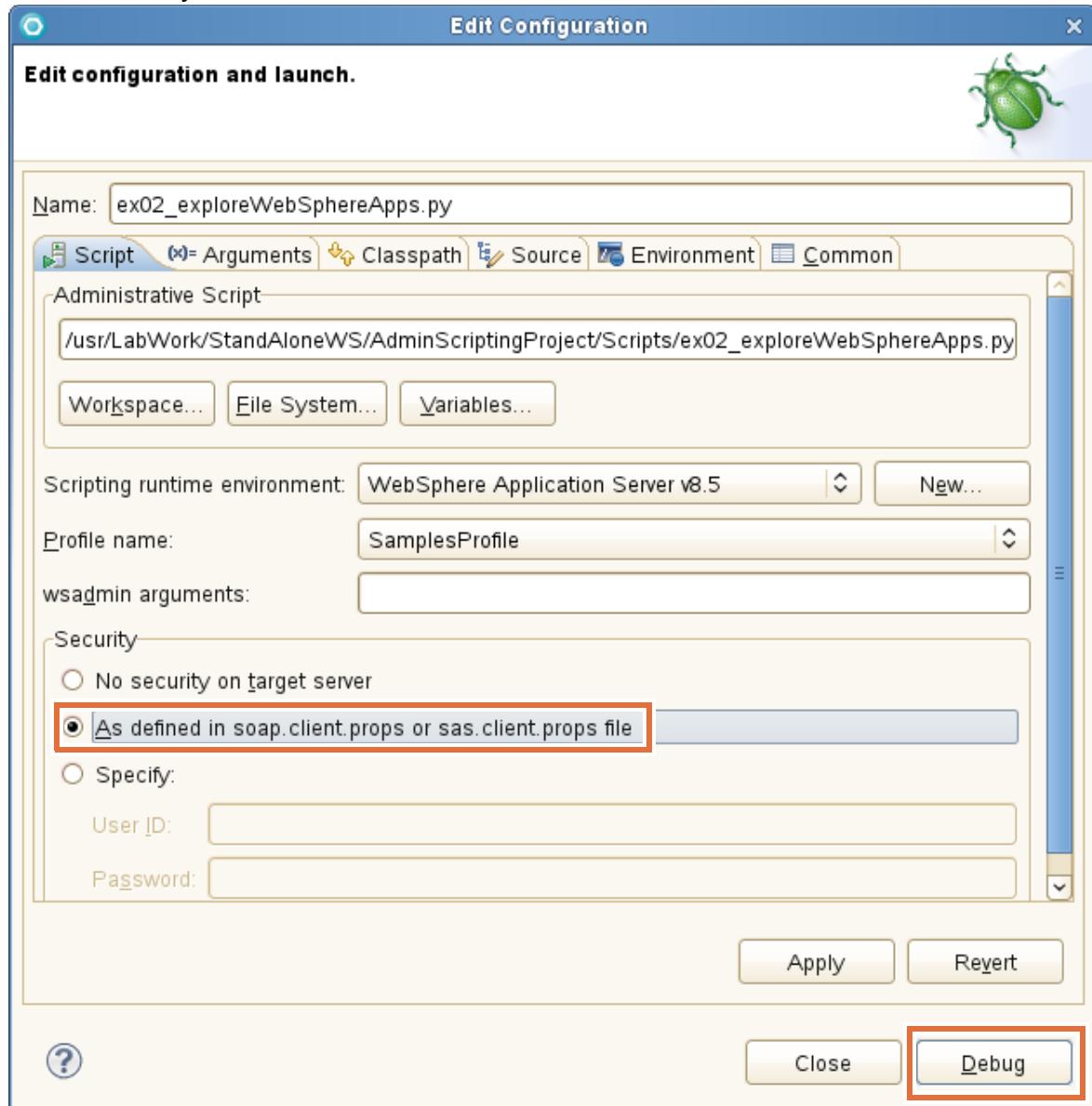
The **Breakpoints** view displays a list of all breakpoints in all projects in the workbench. A breakpoint entry indicates the line and file where it is located. The check box beside each entry controls whether the breakpoint is enabled or disabled.

- \_\_\_ 3. Run the `ex02_exploreWebSphereApps.py` script with the Jython debugger.
- \_\_\_ a. Right-click anywhere in the `ex02_exploreWebSphereApps.py` editor view and select **Debug As > Administrative Script**



- \_\_\_ b. In the **Edit configuration and launch** dialog:

- Select **WebSphere Application Server v8.5** from the Scripting runtime environment list.
- Select **SamplesProfile** from the Profile name list.
- Select **As defined in soap.client.props or sas.client.props file** in the Security section.



- Click **Debug**. The script is started in debug mode, and execution suspends on the first breakpoint encountered.



## Information

The Jython debugger takes some time to start as it connects to the WebSphere Application Server. Watch for the message **Starting the Jython debugger** on the status line (bottom right corner of the workbench). When this message no longer displays, the script is suspended at the first breakpoint, and the stack frames are displayed in the Debug view.

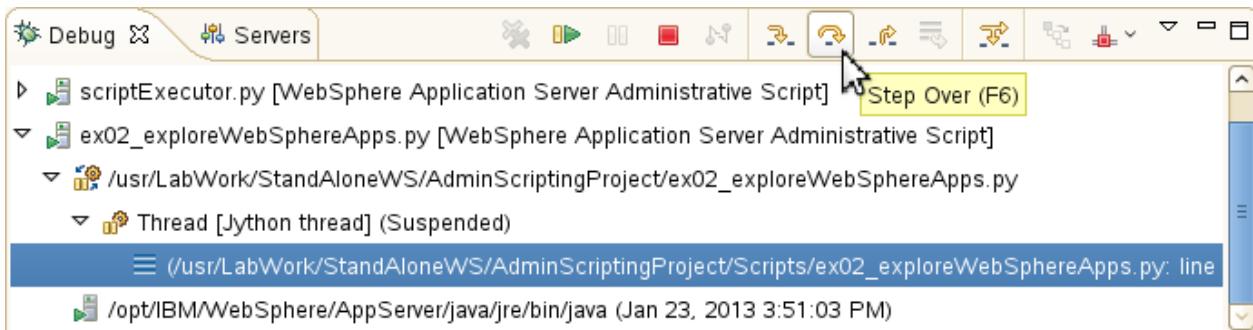
When you use the Jython debugger for the first time, the debugger builds an index of runtime jars and caches the index in a predefined directory. You see messages that begin with **sys-package-mgr\*: processing new jar** in the Console view. This caching operation is run only once per WebSphere Application Server run time.

- 4. Next, step through the code in the `ex02_exploreWebSphereApps.py` script by using the various stepping options that the debugger provides.
  - a. The code execution stops at the first breakpoint that is encountered. Look at the line that is highlighted in the Editor view to determine where the line is. In this case, it is the following line:

```
for i in range (len(appList)):
```

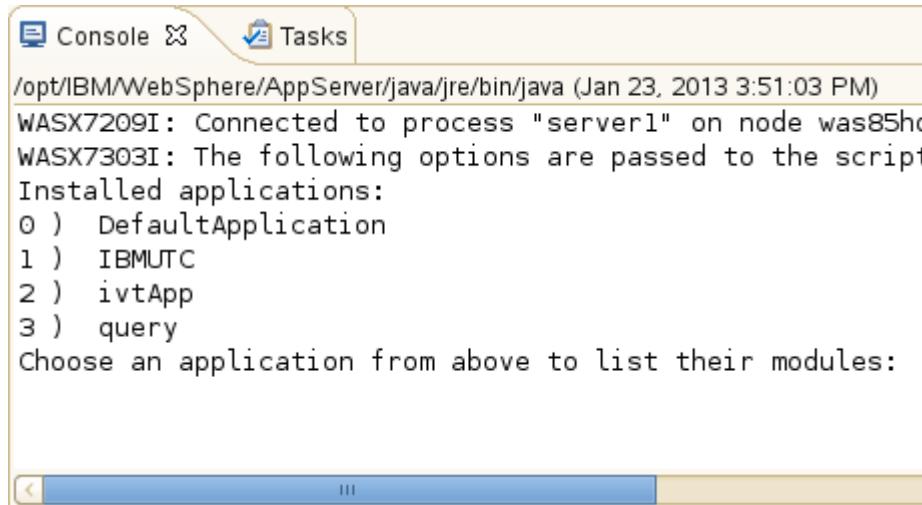
```
py ex02_exploreWebSphereApps.py &
2 import os
3 def isDigit(value):
4     if str(value).isdigit() != 1:
5         return "false"
6     return "true"
7
8 appList=AdminApp.list().split(lineSeparator)
9
10 while 1:
11     print 'Installed applications:'
12     for i in range (len(appList)):
13         print i, ') ', appList[i]
14
```

- \_\_\_ b. Run the next method invocation. In the **Debug** view, click **Step Over**. Alternatively, you can press the **F6** key.



The **Step Over** or **F6** operation runs the next method call at the current line without entering its code and then suspends execution. It is useful when stepping over methods that are already debugged.

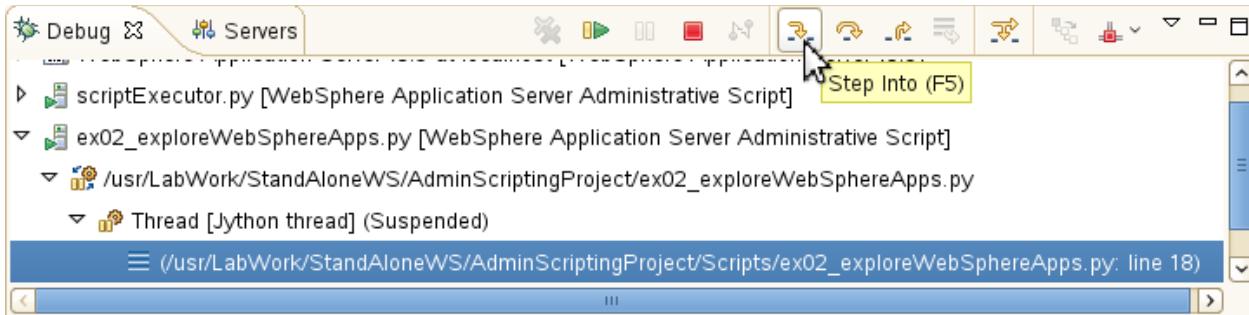
- \_\_\_ c. Continue clicking **Step Over** until the `for` loop completes its execution. The loop completion is indicated when Console view shows the installed applications and displays the prompt for input. As you step over each method, notice how the editor view progressively highlights the line that you are running.



- \_\_\_ d. In the Console view, type `0` to select the **DefaultApplication** and press Enter. Execution resumes then stops at the breakpoint that is set at the following line of code:

```
if isDigit(inputValue) == "true" :
```

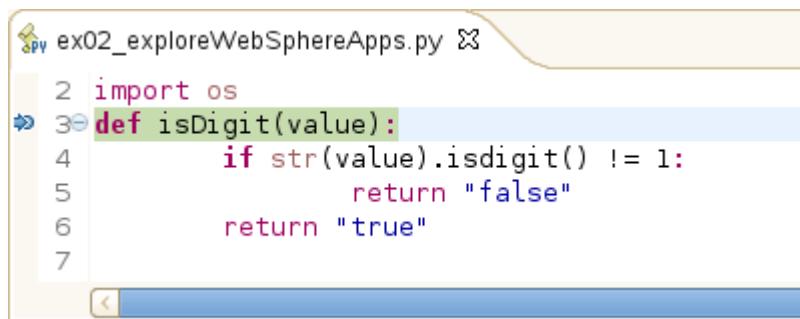
- \_\_\_ e. This line runs the `isDigit()` function that you defined in the script. To step through the code for the function, use the Step Into operation. In the **Debug** view, click **Step Into** or press the **F5** key.



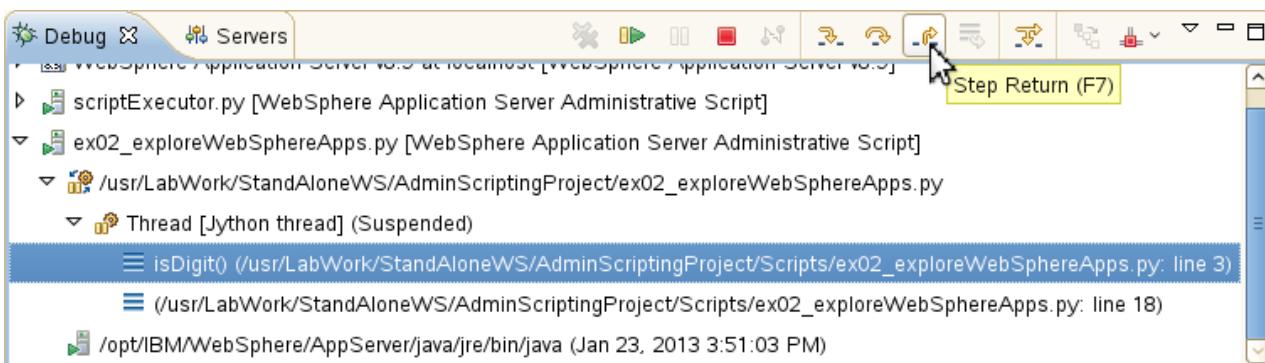
The **Step Into** or **F5** operation steps into a method call and suspends execution on the first line in the method. This operation is useful when you must step through the code of a method that you are starting.

In your case, execution stops at the following line of code since you put a breakpoint on the function definition line:

```
def isDigit(value) :
```

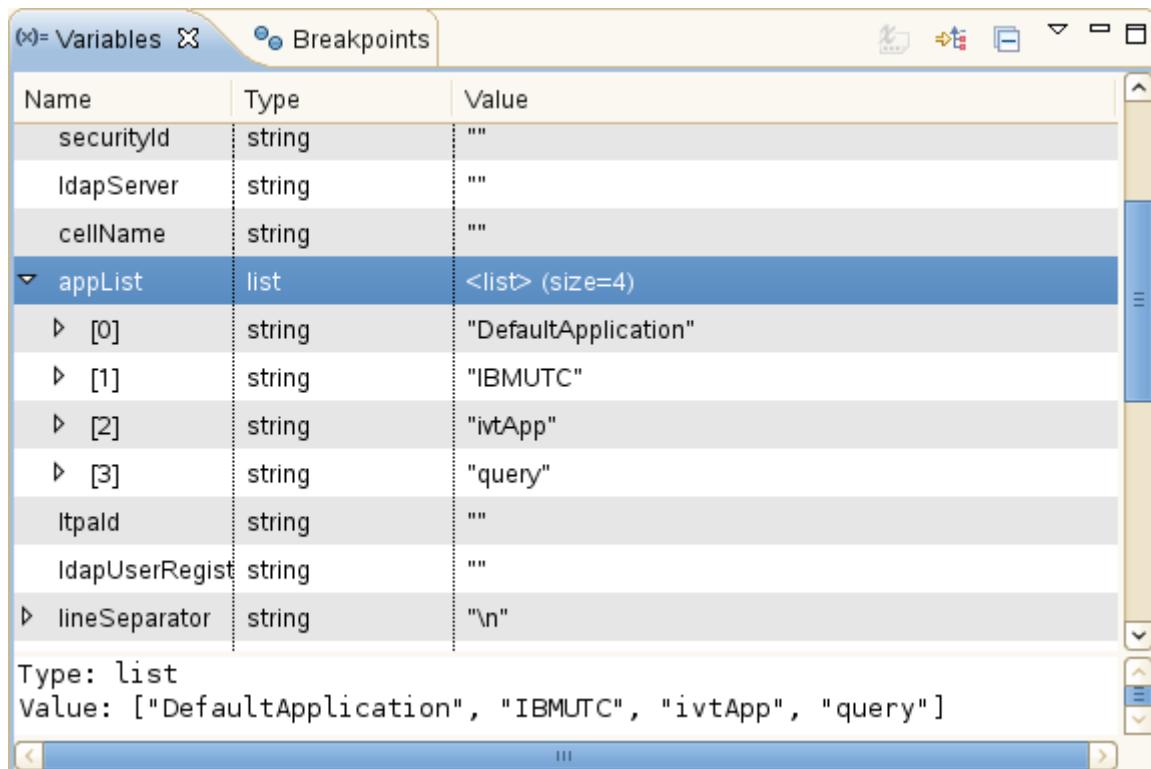


- \_\_\_ f. Finally, run the entire function until the `return` statement is encountered. In the **Debug** view, click **Step Return** or press the **F7** key.



If the **Step Return** or **F7** operation is issued from inside a function, execution resumes until a `return` statement is encountered, at which point execution is suspended. The return value can then be seen in the **Variables** view.

- \_\_\_ 5. Next, look at the **Variables** view to see the current values of the variables that the script creates.
  - \_\_\_ a. Double-click the **Variables** tab to gain focus on the Variables view and maximize it. A list of all the Jython variables that the script created, as well as variables created by the run time during its execution, is displayed. For each variable, the Variables view displays its name, type and value.
  - \_\_\_ b. In the Variables view, expand **appList**. The elements of the **appList** list are displayed along with their current values.



Name	Type	Value
securityId	string	""
ldapServer	string	""
cellName	string	""
appList	list	<list> (size=4)
▷ [0]	string	"DefaultApplication"
▷ [1]	string	"IBMMUTC"
▷ [2]	string	"ivtApp"
▷ [3]	string	"query"
ltpald	string	""
ldapUserRegist	string	""
lineSeparator	string	"\n"

Type: list  
 Value: ["DefaultApplication", "IBMMUTC", "ivtApp", "query"]

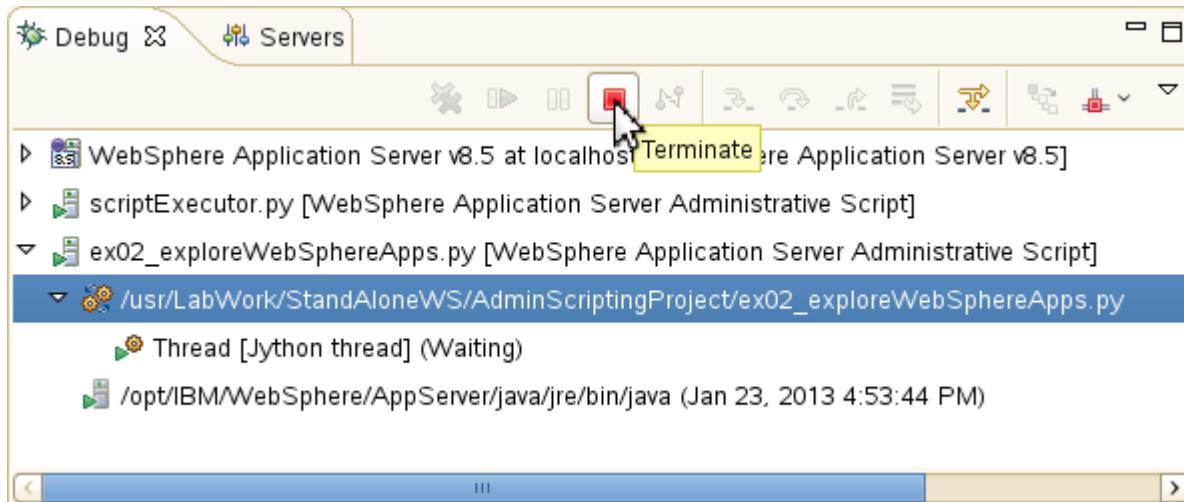


### Information

Jython variable values in the debugger are read-only. You cannot modify the value of a Jython variable in the debugger.

- \_\_\_ c. Scroll through the list to examine the value of other variables. When finished, restore the Variables view to its original size by double-clicking its tab.

6. Finally, end the debug session. In the **Debug** view, click **Terminate**.



The script stops at the last point of execution. If the script is rerun in debug mode, the breakpoints settings are still in effect.

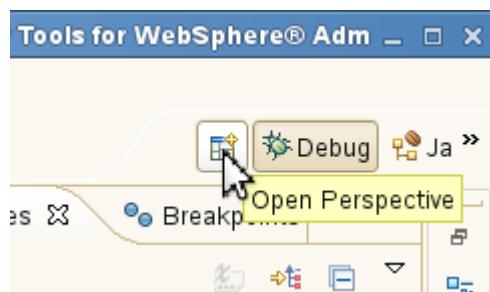
You used the IADT debugger tool to step through Jython script code and view variable values.

## Section 6: Using the administrative console command assistance to help develop a jython script

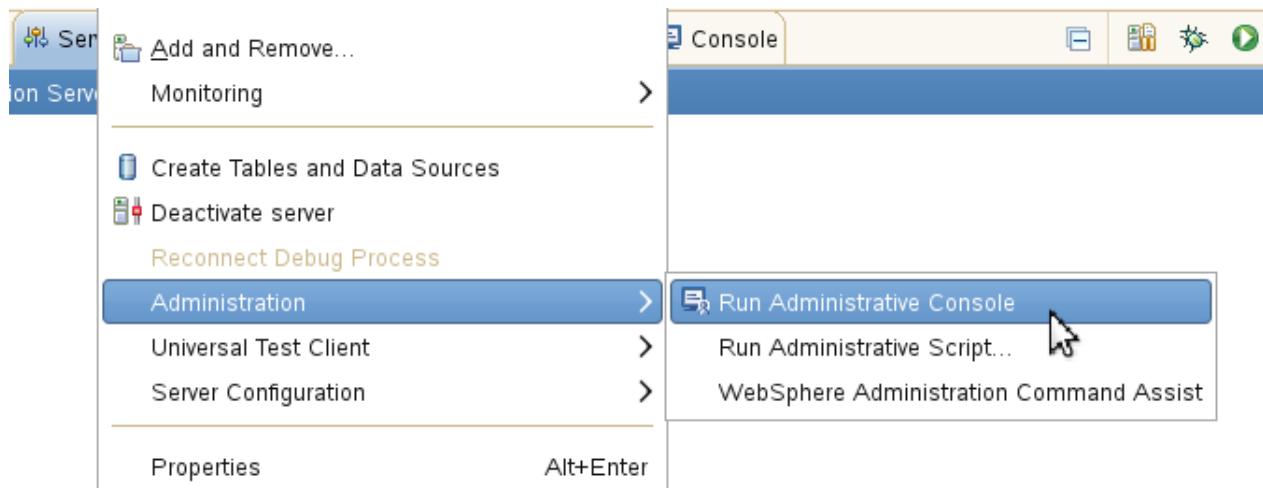
The Jython development environment that IADT provides can capture the notifications that the administrative console command assistance tools are generates. These notifications contain the Jython syntax of actual commands that are run in the administrative console to perform administrative tasks. Capturing these commands can expedite the development of administrative Jython scripts, as they can easily be inserted into a script opened in the Jython editor for reuse.

In this section, you enable the **Command assistance notifications** feature of the administrative console and set up the IADT to monitor notifications. You then use the console to query the JDBC providers and a data source that is defined in the environment. You can then capture the resulting notifications in the IADT and insert them into a new script.

- \_\_\_ 1. Log in to the WebSphere administrative console.
- \_\_\_ a. Return to the J2EE Perspective. Click **Open Perspective** and open the **J2EE** perspective.



- \_\_\_ b. From the **Servers** view, right-click **WebSphere Application Server v8.5 at localhost** and select **Administration > Run administrative console**



- \_\_ c. Click the **OK** button at the Invalid Certificate window.



- \_\_ d. The administrative console login page is opened in the Admin Console view. In the user ID field, type **wasadmin**, and in the Password field, type **websphere**



- \_\_ e. Click **Log in**.



## Troubleshooting

### Logging in to the WebSphere Integrated Solutions Console (Admin Console)

For the VMware image used in this course, you might experience the following problem when you try to log in: The **User ID** field is unresponsive, and you cannot type the ID.

If you experience this problem, try one of the following work-arounds:

1. Close the console by clicking the x on the Admin Console tab, and perform the steps to run the administrative console again.
2. Click the **Log in** button without typing the **User ID**. You see the message: “Invalid User ID or password.” Then, try entering the ID and password again.
3. Use an external web browser to access the administrative console. Use the web address <http://was85host:9060/ibm/console>

- 
4. Enable the **command assistance notifications** feature. You enable this feature on a Help portlet for an action where command assistance is available, for example, when listing JDBC providers.
    - a. In the navigation pane, select **Resources > JDBC > JDBC Providers**.
    - b. Notice the Help portlet on the right side of the administrative console. Under Command Assistance, click the **View Administrative scripting command for last action** link.

**JDBC providers**

**JDBC providers**

Use this page to edit properties of a JDBC provider. The JDBC provider object encapsulates the specific JDBC driver implementation class for access to the specific vendor database of your environment. Learn more about this task in a [guided activity](#). A guided activity provides a list of task steps and more general information about the topic.

Scope: **All scopes**

Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, [see the scope settings help](#).

All scopes

**Preferences**

New... Delete

Select	Name	Scope	Description
<input type="checkbox"/>	<a href="#">Derby JDBC Provider</a>	Node=was85hostNode01,Server=server1	Derby embedded non-XA JDBC Provider

Total 1

**Help**

**Field help**  
For field help information, select a field label or list marker when the help cursor is displayed.

**Page help**  
[More information about this page](#)

**Command Assistance**  
[View administrative scripting command for last action](#)



## Information

Not all administrative console actions have wsadmin commands that are directly associated with them. If a command assistance link is listed in the Help portlet, wsadmin commands exist for the last console action that you completed, and command assistance is available for that action.

If, however, the Help portlet on the right side of the administrative console panel does not have a command assistance link in it, no command assistance data is available for the last console action.

The Administrative Scripting Commands window opens in a web browser and displays the Jython command syntax for the last run console action (list JDBC providers for cell).

Administrative Scripting Commands

The wsadmin scripting commands that map to actions on the administrative console display in the Jython language.

Preferences

**Administrative Scripting Command**

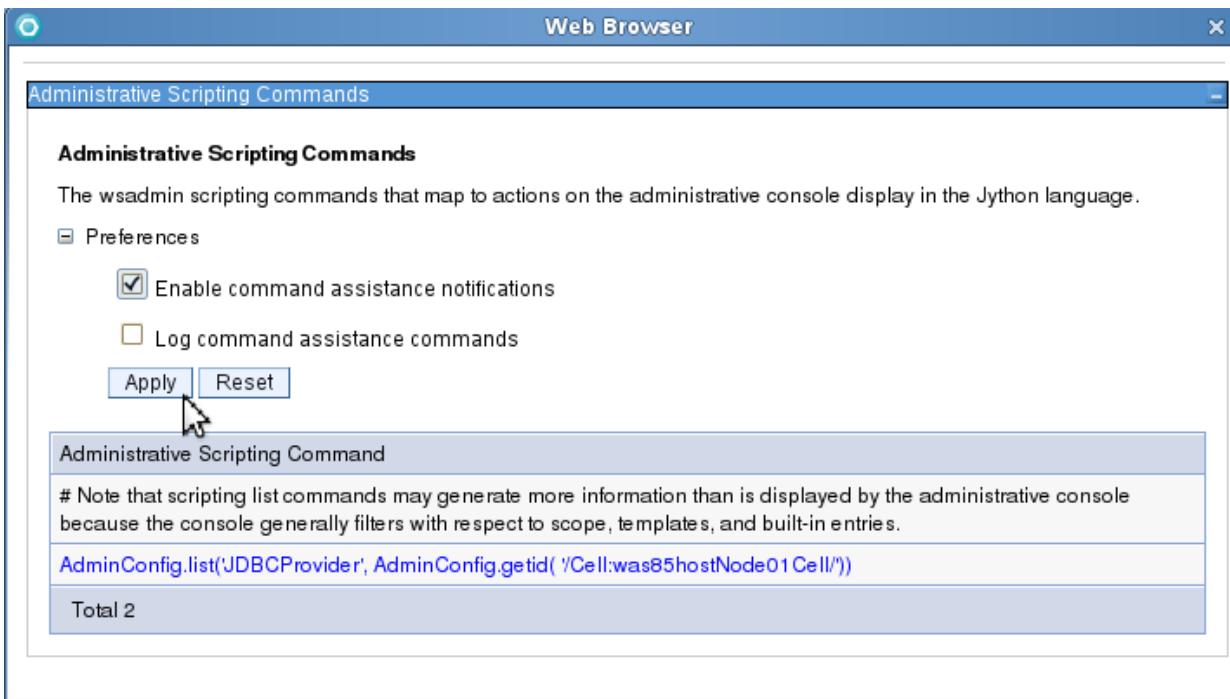
```
# Note that scripting list commands may generate more information than is displayed by the administrative console because the console generally filters with respect to scope, templates, and built-in entries.
```

```
AdminConfig.list('JDBCProvider', AdminConfig.getid( '/Cell:was85hostNode01 Cell'))
```

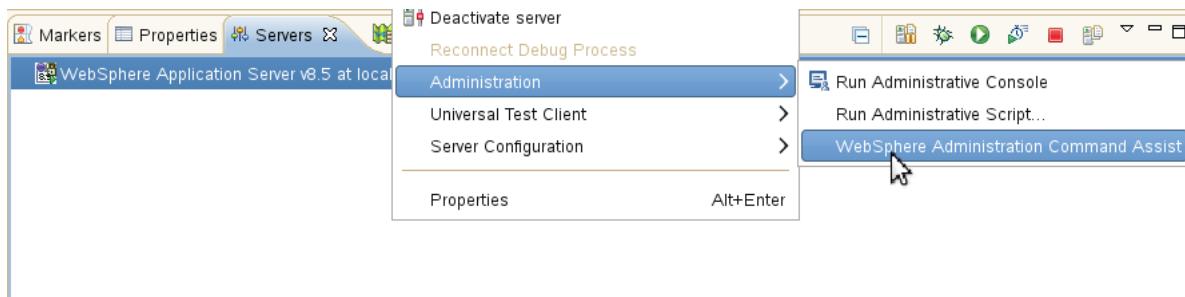
Total 2

- \_\_\_ c. In the Administrative Scripting Commands window, expand **Preferences**.

- \_\_\_ d. Select the **Enable command assistance notifications** check box.

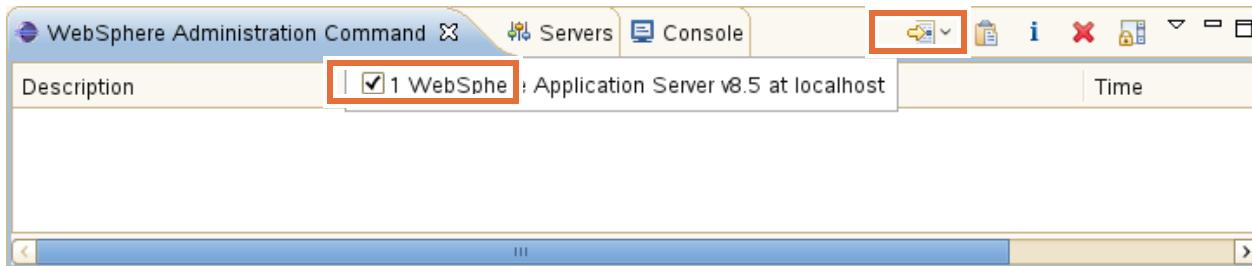


- \_\_\_ e. Click **Apply**.
- \_\_\_ f. Close the **Administrative Scripting Commands** window.
- \_\_\_ 5. Next, set up the IADT to monitor command assistance notifications.
- \_\_\_ a. In the **Servers** view, right-click **WebSphere Application Server v8.5 at localhost** and select **Administration > WebSphere Administration Command Assist**.



The WebSphere Administration Command view opens.

- \_\_\_ b. Click the down arrow beside **Select Server to Monitor** and select (click the check box) **WebSphere Application Server v8.5**.



The IADT now listens to notifications that come from the SamplesProfile server.



#### Note

The server that you want to monitor must be started; otherwise it is disabled in the **Select Server to Monitor** selection list.

- \_\_\_ 6. You can now perform operations in the administrative console and capture the generated commands in IADT. For example, display the data sources that are defined under the **Derby JDBC provider**.
- In the Admin Console view, select **Resources > JDBC > JDBC Providers**.
  - In the JDBC provider pane, scroll-down and click the **Derby JDBC Provider** link.
  - Click the **Data sources** link located under **Additional Properties**. The data sources for the Derby JDBC provider are listed.
  - Now look at the **WebSphere Administration Command** view.

WebSphere Administration Command		
Description	Command	Time
WASX7056I: Method: list	AdminConfig.list('JDBCProvider', AdminConfig.getid('/Cell:was85hostNode01Cell'))	1/23/13 8:14 PM
WASX7056I: Method: list	AdminConfig.list('DataSource', AdminConfig.getid('/Cell:was85hostNode01Cell/Node'))	1/23/13 8:14 PM

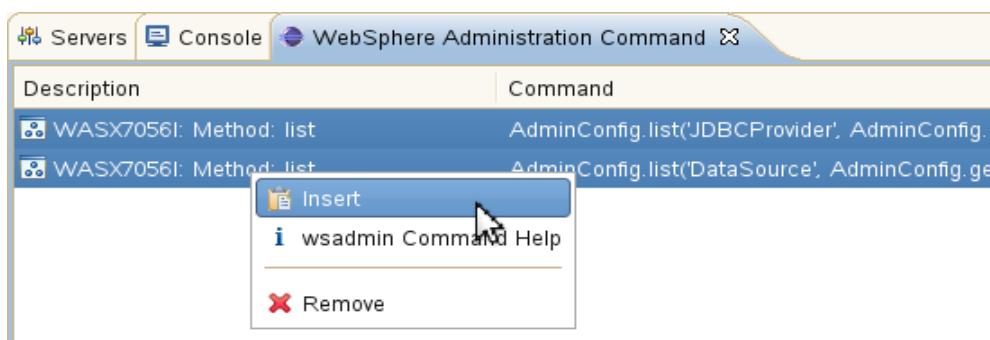
The commands that are used to display the list of JDBC Providers and the Default data sources are captured and are shown in the view.

- \_\_\_ 7. Next, use the recorded commands in a Jython script.

**Important**

The recorded commands can be inserted only into an open Jython editor.

- \_\_\_ a. Create another Jython script named **ex02\_listJDBCResources.py** in the Scripts folder.
- \_\_\_ b. In the **ex02\_listJDBCResources.py** editor view, click the first blank line after the comment lines to position the cursor there.
- \_\_\_ c. In the WebSphere Administration Command view, group-select the listed commands (hold the Shift key and select all commands).
- \_\_\_ d. Right-click the selection and select **Insert**



The two commands are copied into the **ex02\_listJDBCResources.py** editor view.

- \_\_\_ e. Add a **print** statement in front of the **AdminConfig()** methods to improve the formatting.

```

1 #
2 # TODO: enter JYTHON code and save
3 "
4 print AdminConfig.list(
5   'JDBCProvider', AdminConfig.getid(
6     '/Cell:was85hostNode01Cell/'))
7 print AdminConfig.list(
8   'datasource', AdminConfig.getid(
9     '/Cell:was85hostNode01Cell/Node:was85hostNode01/Server:server1/JDBCProvider:Derby JDBC Provider/'))

```

- \_\_\_ f. Type **Ctrl+S** to save your changes.
- \_\_\_ 8. Start the scriptExecutor and run the **ex02\_listJDBCResources.py** script.
  - \_\_\_ a. Run the scriptExecutor utility script. In the Enterprise Explorer view, expand **AdminScriptingProject > Utilities**. Right-click **scriptExecutor.py** and select **Run As > Administrative Script**.

- \_\_ b. Run the **ex02\_listJDBCResources.py** script. In the scriptExecutor Console view, enter:

## ex02 listJDBCResources.py

The resulting script displays the installed Derby JDBC Providers and the default data source.

## ***Section 7: Cleaning up the environment***

- \_\_\_ 1. Stop the scriptExecutor script. In the scriptExecutor Console window, enter: `quit`.
  - \_\_\_ 2. Stop the server.
    - \_\_\_ a. Display the Servers view by clicking the **Servers** tab.
    - \_\_\_ b. Make sure the **WebSphere Application Server v8.5 at localhost** is selected and click **Stop the server**.



Wait until the Servers view displays a status of “Stopped”.

3. Exit the IADT by selecting **File > Exit**.

## End of exercise

## Exercise review and wrap-up

In the exercise, you learned the basic constructs of the Jython language and used the IBM Assembly and Deploy Tools to develop Jython scripts. You started by creating a workspace and Jython project, and defining the target server for your script execution. You then created and ran a simple “Hello world” Jython script. To explore basic Jython language constructs, you imported and ran sample scripts, and then developed an administrative script. You also familiarized yourself with the Jython debugger and used the administrative console command assistance to help develop a Jython script.



# Exercise 3. Using the Help and AdminConfig objects

## What this exercise is about

In this exercise, you learn how to use the AdminConfig administrative object by writing scripts to create, query, modify, and delete a configuration object. Following the general approach for completing a configuration task by using the AdminConfig object, you gain familiarity with the primary methods and resources that are required to effectively use it.

## What you should be able to do

At the end of this exercise, you should be able to:

- Use the Help object to get help on the administrative objects
- Apply the general steps that are required to complete a configuration task by using the AdminConfig object
- Employ the primary methods and resources that help in performing each step
- Use the AdminConfig object to query, create, modify, and delete a configuration object

## Introduction

The AdminConfig administrative object is used to manage the configuration information that is stored in the WebSphere Application Server configuration repository. It can be used to query, create, modify, and delete configuration objects.

A *Virtual Host* is one such configuration object. You can use virtual hosts to present to clients a single application server on a single machine or as multiple application servers each on their own host machine. You can separate and control which resources are available for client requests by combining multiple host machines into a single virtual host, or by assigning host machines to different virtual hosts.

In this exercise, you create, query, modify, and delete a virtual host on the *SamplesProfile server1* application server by using the AdminConfig object. A virtual host object provides a good context for learning how to use the AdminConfig object since configuring one is relatively simple and has little impact on other configuration objects.

As such, it allows you to concentrate on how to use the AdminConfig object.

The general approach for completing a configuration task with the AdminConfig object is depicted in Figure 1. It also shows the methods and resources that can be used to help in performing a particular step.

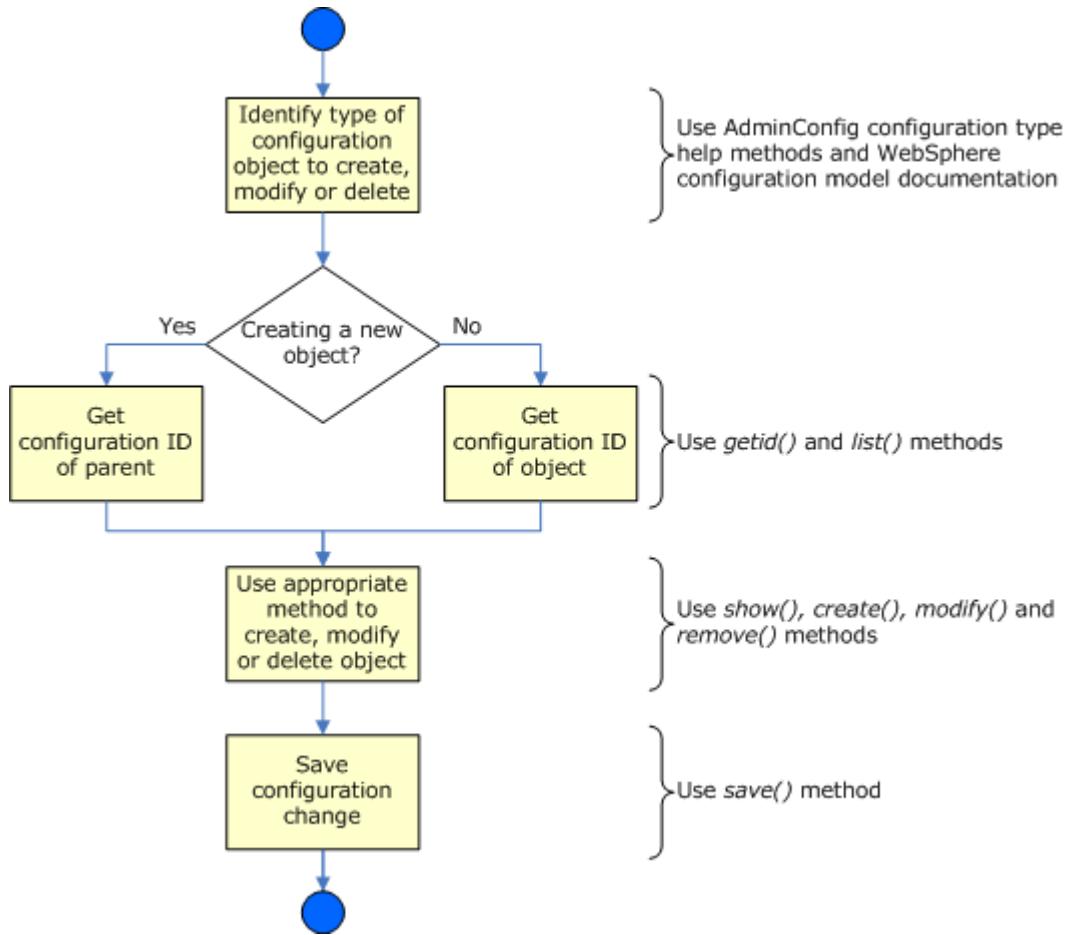


Figure 1 - How to use the AdminConfig object

Using the IBM Assemble and Deploy Tools (IADT), you develop Jython scripts that use these primary methods of the AdminConfig object.

## Requirements

To complete this exercise, the WebSphere Application Server Network Deployment V8.5 product must be installed and a server named server1 must be created in the default profile. You also require the IBM Assembly and Deploy Tools for WebSphere Administration 8.5 (IADT)

## Exercise instructions



### Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

`<was_root>`: /opt/IBM/WebSphere/AppServer

`<profile_root>`: /opt/IBM/WebSphere/AppServer/profiles

### Section 1: Preparing the environment

In this step, you open the same IBM Assembly and Deploy Tools (IADT) workspace that you used in the previous exercise, namely, *StandAloneWS*. This workspace already contains a Jython project definition and the definition of the *SamplesProfile server1* stand-alone server. It also already includes the utility script, *scriptExecutor.py*, which is used to facilitate script execution inside the IADT.



### Note

If you must create or re-create the *StandAloneWS* workspace from scratch, following the steps that are described in Section 1 Preparing the environment of Exercise 2.

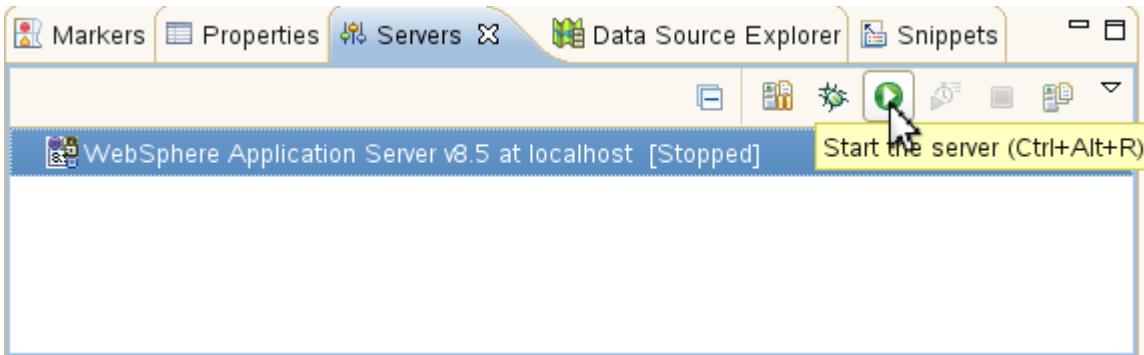
- \_\_\_ 1. Start the IBM Assembly and Deploy Tools and open the **StandAloneWS** workspace.
  - \_\_\_ a. On the desktop, double-click the **IBM Assembly and Deploy Tools** icon.
  - \_\_\_ b. In the Workspace Launcher window, make sure that the Workspace field has a value of **/usr/LabWork/StandAloneWS**.
  - \_\_\_ c. Click **OK**. The IADT opens and displays the Java EE perspective.
- \_\_\_ 2. Start the **SamplesProfile server1** server.



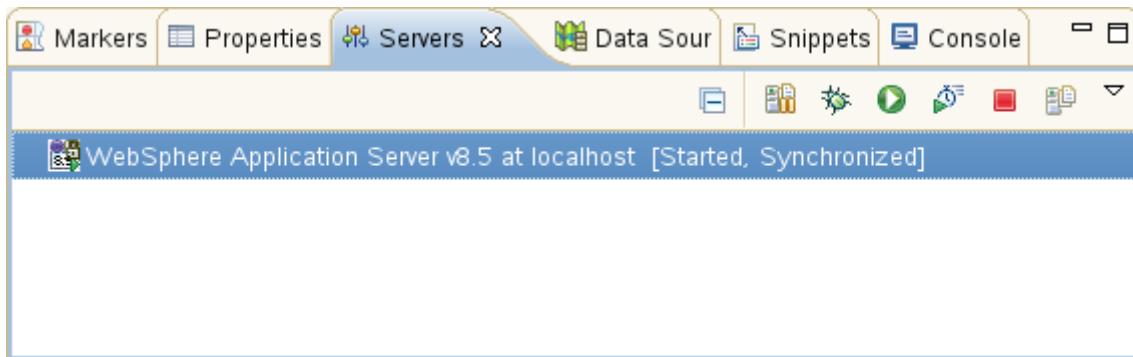
## Information

Using the AdminConfig object does not require a connection to a running server. Therefore, the scripts that are developed in this lab can run without being connected to *SamplesProfile server1*. However, because you later open the administrative console to verify your configuration changes, you are asked to start the server now for convenience.

- \_\_ a. In the Servers view, make sure **WebSphere Application Server v8.5 at localhost** is selected and click **Start the server**.

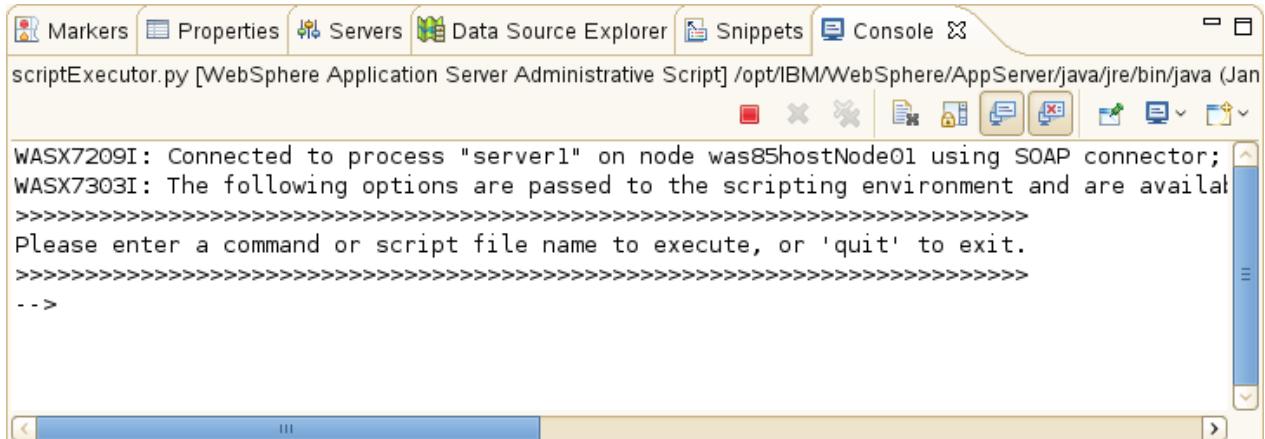


- \_\_ b. Wait until the server is started. As the server starts, focus is switched to the **Console** view where startup messages are displayed. When the server is started, the Servers view displays a status of **Started**.



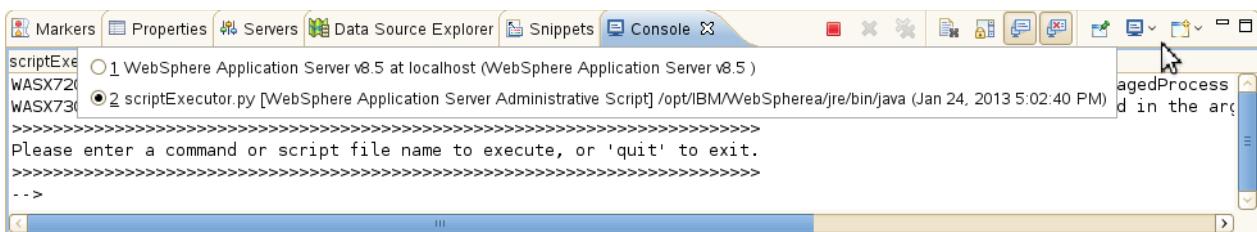
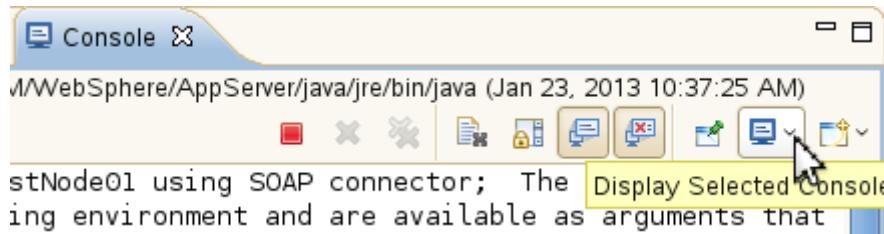
- \_\_ 3. Run the scriptExecutor utility script.
  - \_\_ a. In the Enterprise Explorer view, expand **AdminScriptingProject > Utilities**. Right-click **scriptExecutor.py** and select **Run As > Administrative Script**.
  - \_\_ b. After a few moments, the **Console** view displays a message that confirms the establishment of a wsadmin connection with the *server1* server. An input prompt

is also displayed for running an administrative command or a script file. You are now ready to develop and run your scripts.



## Information

Observe that you have two Console views opened: one for the *scriptExecutor* script and the other of the *server1* standard output. To switch between Console views, click the drop-down arrow of the **Display Selected Console** button and select the console output to view.



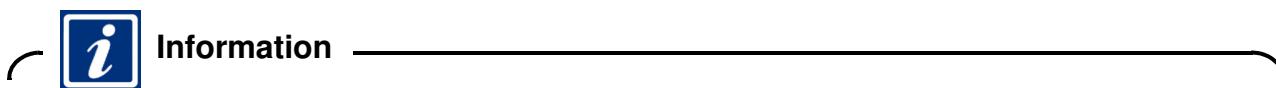
## ***Section 2: Identifying and exploring the configuration type of a virtual host object***

The first step in using the AdminConfig object is to identify the type of the target configuration object in order to discover its attribute names and types. This information is required if you are creating an object or modifying an existing one because you must supply the appropriate values for the attributes of the object. Also, if you are deleting a configuration object, you still must know its type to obtain its configuration ID.

In this part of the exercise, you use several AdminConfig configuration type help methods and the WebSphere configuration model documentation to identify and explore the type that corresponds to a virtual host object.

- \_\_\_ 1. What is the name of the configuration type for a virtual host object? Start by listing the names of all WebSphere configuration types by using the `types()` method. In the scriptExecutor Console view, enter:

```
print AdminConfig.types()
```



When you gain focus on the scriptExecutor Console view, the cursor is not immediately positioned on the input prompt line. However, you can start typing a command and the cursor is automatically repositioned on the input prompt line.

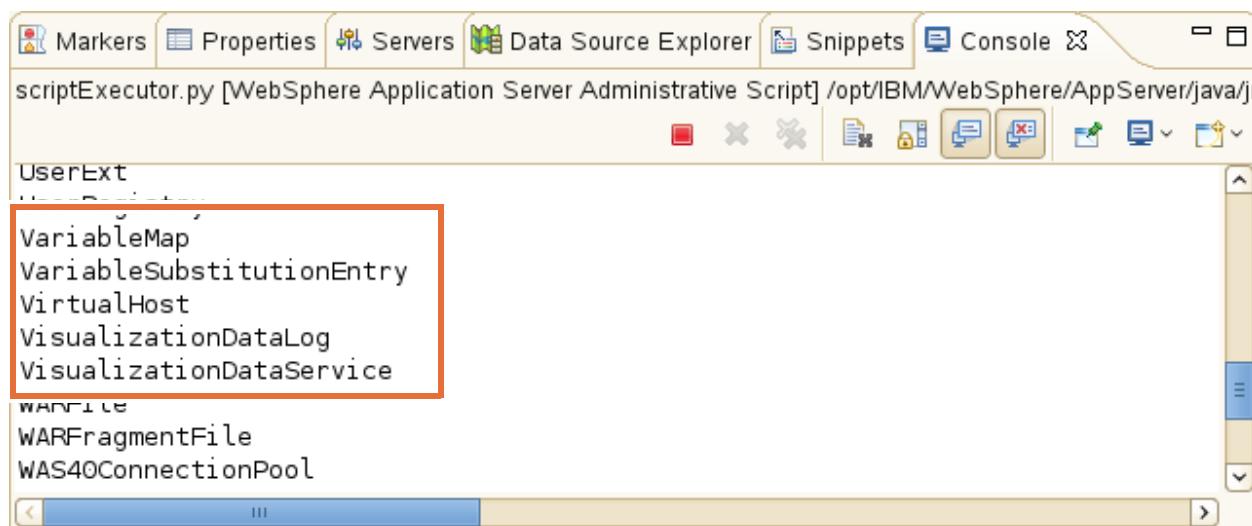
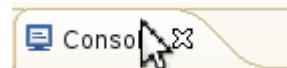
- \_\_ 2. The types method returns an alphabetical list of over 560 configuration types! Which one corresponds to a virtual host object? This result is where some intuitive thinking

helps. Might it be that a type named **VirtualHost** exists? To find out, scroll-down the Console view until you see the type names that start with a 'v'.



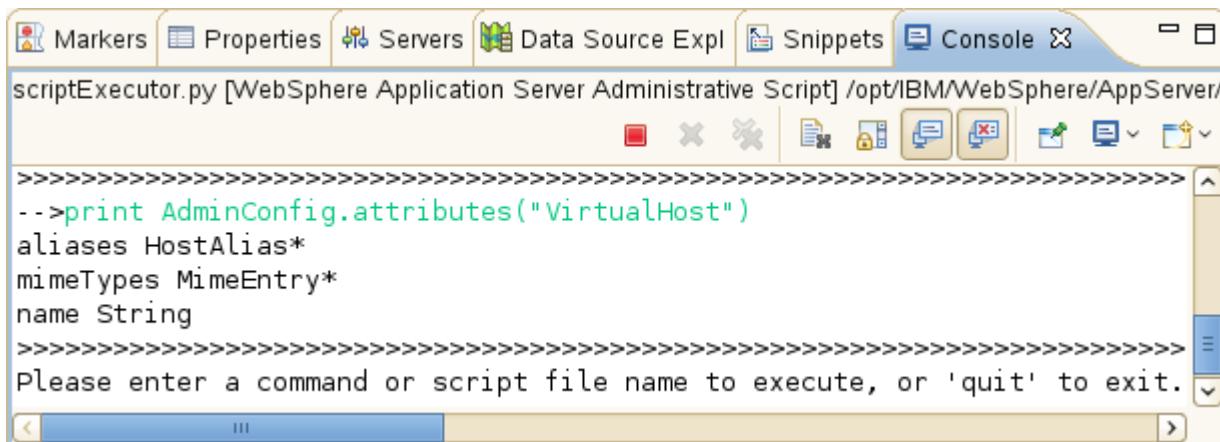
### Hint

You can maximize any view by double-clicking its tab label. Conversely, to restore it to its original size, double-click the tab label again.



3. Now that you identified the **VirtualHost** configuration type, what attributes does it have? To find out, use the `attributes (aType)` method. In the `scriptExecutor` Console view, enter:

```
print AdminConfig.attributes("VirtualHost")
```



The VirtualHost type has three attributes named `aliases`, `mimeTypes`, and `name`. The first two are themselves configuration objects. In fact, the asterisk notation indicates that they are collections of configuration objects. The last attribute is a simple string.

- \_\_\_ 4. What does each attribute represent? A look at the WebSphere configuration model documentation helps answer this question.
  - \_\_\_ a. Open a file system browser such as Nautilus, and go to `<was_root>/web/configDocs`.
  - \_\_\_ b. Double-click `index.html` to open the home page of the documentation.

**WebSphere Application Model Documentation**

[Home](#) | [All Packages](#) | **All Classes**

Packages

- [activitysessionejbext](#)
- [activitysessionservice](#)
- [activitysessionwebappext](#)
- [adminservice](#)
- [appcfg](#)
- [appdeployment](#)
- [application](#)
- [applicationbnd](#)
- [applicationclientext](#)
- [applicationext](#)
- [applicationserver](#)
- [appmgtservice](#)
- [appprofileapplicationclientext](#)
- [appprofileapplicationnext](#)
- [appprofilecommonext](#)
- [appprofileejbext](#)
- [appprofileservice](#)
- [appprofilewebappext](#)

**WebSphere Application And Configuration Model Documentation**

This documentation provides a summary of the models used by the WebSphere Application Server version 8.0, including J2EE Application Models, IBM J2EE Bindings and Extensions Models, and Application Server Configuration Models.

This documentation provides a title frame, and leftmost and rightmost primary content frames. The leftmost frame contains either a list of all available model packages, or a list of all available model classes, or a description of a single model package along with a listing of the classes within that package. The rightmost panel contains either this documentation note, or contains the details for a single model class.

Here a **model class** is a model class as defined through Rational Rose. Generally, a model class maps to an implementation java class having the same name, but mapped also to an implementation java package.

Model classes include both object types and data types. Data types include both basic data types, such as `Date` and `Integer` as well as enumerated types.

- \_\_\_ c. In the top frame, click the **All Classes** link to list the configuration types by their class names.

- \_\_\_ d. In the navigation pane on the left, scroll-down until you see the **VirtualHost (host)** class and select it. The details of the class are shown on the right pane.

The screenshot shows a web browser displaying the "WebSphere Application Model Documentation". The left sidebar lists various classes under the package "wscommonhost". One class, "VirtualHost (host)", is highlighted with a red box. The main content area is titled "VirtualHost" and contains the following text:

This type is a class for model objects.

Configuration for enabling a single host machine to resemble multiple host machines. Resources associated with one virtual host cannot share data with resources associated with another virtual host, even if the virtual hosts share the same physical machine.

Each virtual host has a logical name and a list of one or more DNS aliases by which it is known. A DNS alias is the TCP/IP host name and port number used to request a Web module resource (such as a servlet, JSP file, or HTML page). For example, it is the "myhost:80" portion of a client request for <http://myhost:80/servlet/snoop>. When a client request is made, the server name and port number are compared to a list of all known aliases in an effort to locate the correct virtual host and serve the requested resource.

Package: [host](#)

Classifier ID: -1  
Instance class name: \* Unspecified \*  
Instance class: \* Unspecified \*

- \_\_\_ e. Scroll-down the right pane to see a summary (Attributes Summary) and detailed description (Attribute Details) of each attribute of the VirtualHost type.
- \_\_\_ f. Close the browser when you are finished.
- \_\_\_ 5. As indicated by the result of the `attributes()` method and the configuration model documentation, the VirtualHost type has attributes that are themselves configuration objects. You can now use these same resources to learn more about each contained type. For example, to view the attributes of the HostAlias type, enter the following in the Console view:

```
print AdminConfig.attributes("HostAlias")
```

```

Markers Properties Servers Data Source Expl Snippets Console X
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/
Please enter a command or script file name to execute, or 'quit' to exit.
-->print AdminConfig.attributes("HostAlias")
hostname String
port String

```

You can also use the links in the configuration model documentation to go to the details page of contained types.

6. Which attributes are required for the `VirtualHost` type? The `AdminConfig` object provides a method called `required()` to help answer such a question. In the `scriptExecutor` Console view, enter:

```
print AdminConfig.required("VirtualHost")
```

```

-->print AdminConfig.required("VirtualHost")
Attribute          Type
name              String

```

As shown in the method results, only the `name` attribute is required for the `VirtualHost` type.



### Note

The `required()` method indicates that `name` is a required attribute while the WebSphere configuration model documentation says that it is not (see *Attribute Details* section of screen capture on previous page). Unfortunately, conflicts such as this one sometimes occur between the different help resources, and determining the correct answer requires experimentation on your part.

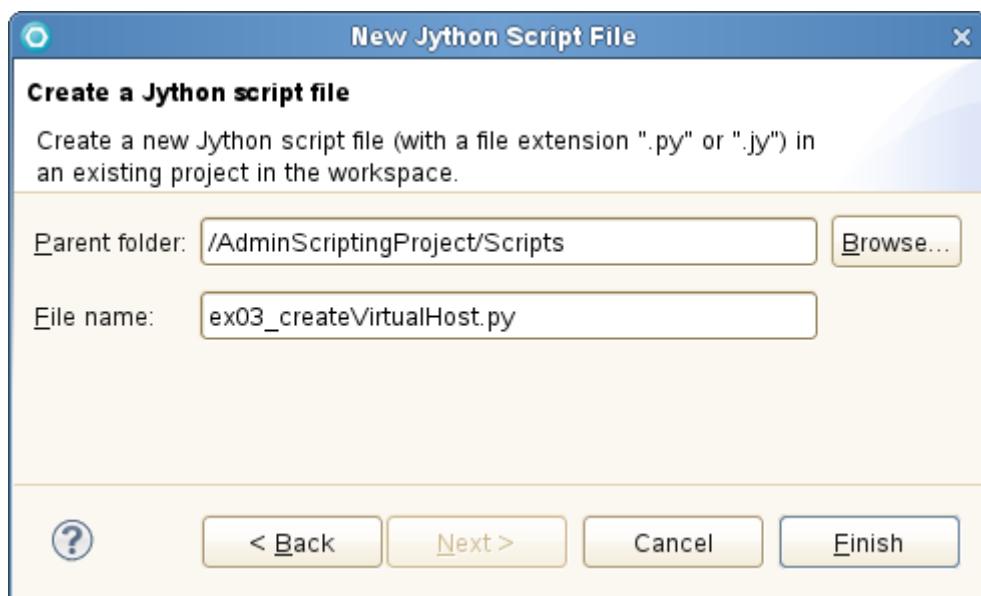
You identified the configuration type of the object that you want to create, `VirtualHost`, and explored its structure and attribute requirements.

### Section 3: Creating a virtual host configuration object

The next step is to create a new Virtual Host object. For the sake of simplicity and to illustrate the most basic way to create a configuration object, the new virtual host is created by specifying its single required attribute: name. Later, you modify it to add more attribute values.

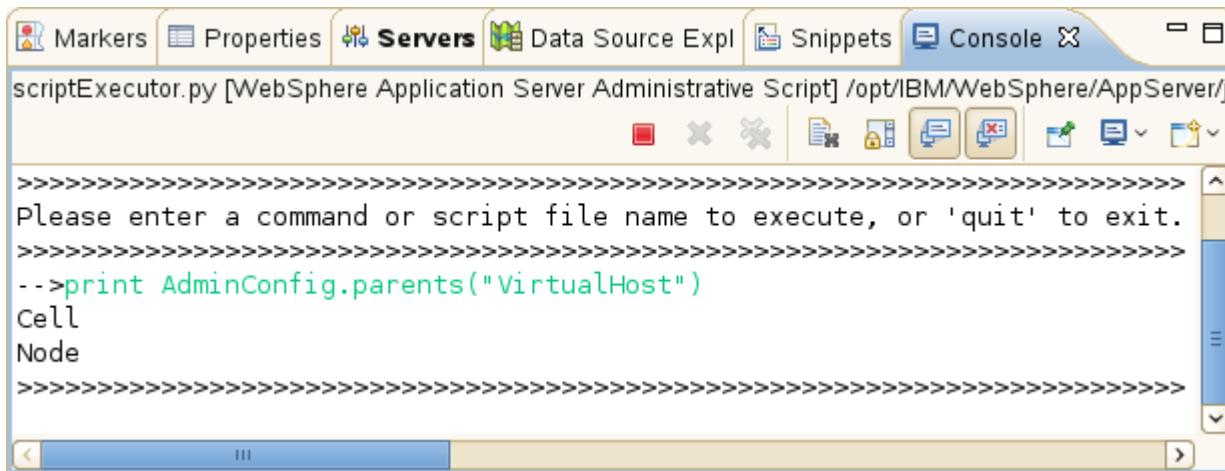
As shown in Figure 1, the steps for creating a configuration object are:

1. Get the configuration ID of a parent object.
  2. Construct and run the appropriate create method.
  3. Save the configuration change.
- 1. First, create a script file named `ex03_createVirtualHost.py` in the Scripts folder to hold the commands that you develop for creating a new virtual host.
    - a. In the Enterprise Explorer view, right-click the **Scripts** folder and select **New > Other**. The Select a wizard dialog opens.
    - b. In the **Select a wizard** dialog, expand **Jython** and select **Jython Script File**. Click **Next**.
    - c. In the **Create a Jython script file** dialog, type `ex03_createVirtualHost.py` in the File name field.



- d. Click **Finish**. The new script file is now shown in the Enterprise Explorer view in the Scripts folder, and is opened in the Jython editor view.
- 2. What is the parent of a Virtual Host object? You can use the `parents(aType)` method to find out. In the scriptExecutor Console view, enter:

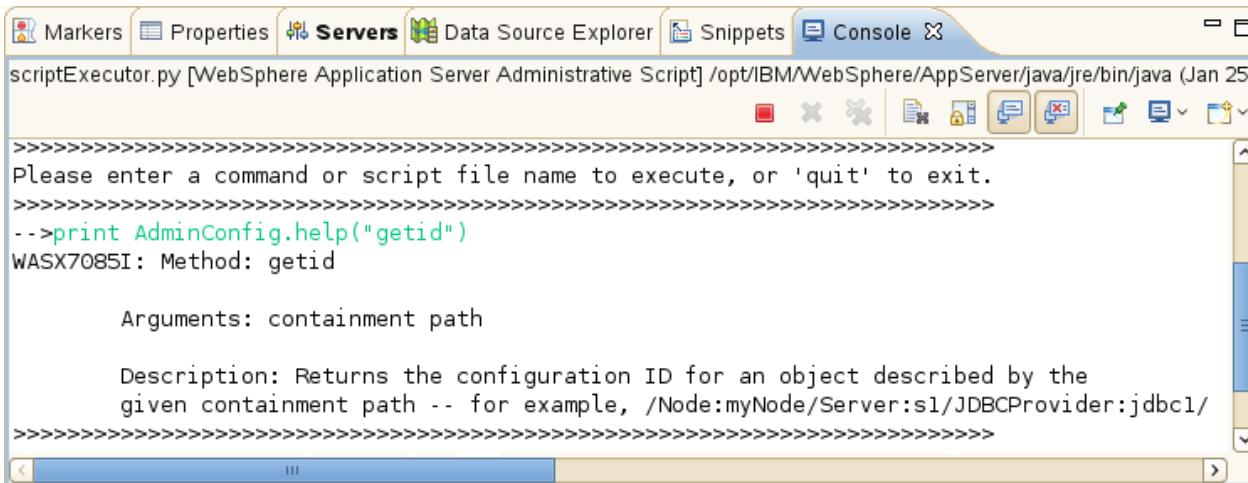
```
print AdminConfig.parents("VirtualHost")
```



The method results show that Cell and Node are parent types for VirtualHost, that is, they can contain a virtual host object. Therefore, you must get the configuration ID of either a Cell or Node object to use as the parent for the new virtual host.

- \_\_\_ 3. Retrieve the configuration ID of a **cell** parent by using the `getid(anObjectContainmentPath)` method. Since the environment is a stand-alone server, it really does not matter whether you attach the new virtual host object to the single cell or node object that is present.
- \_\_\_ a. First, get help on how to start the `getid()` method. In the scriptExecutor Console view, enter:

```
print AdminConfig.help("getid")
```



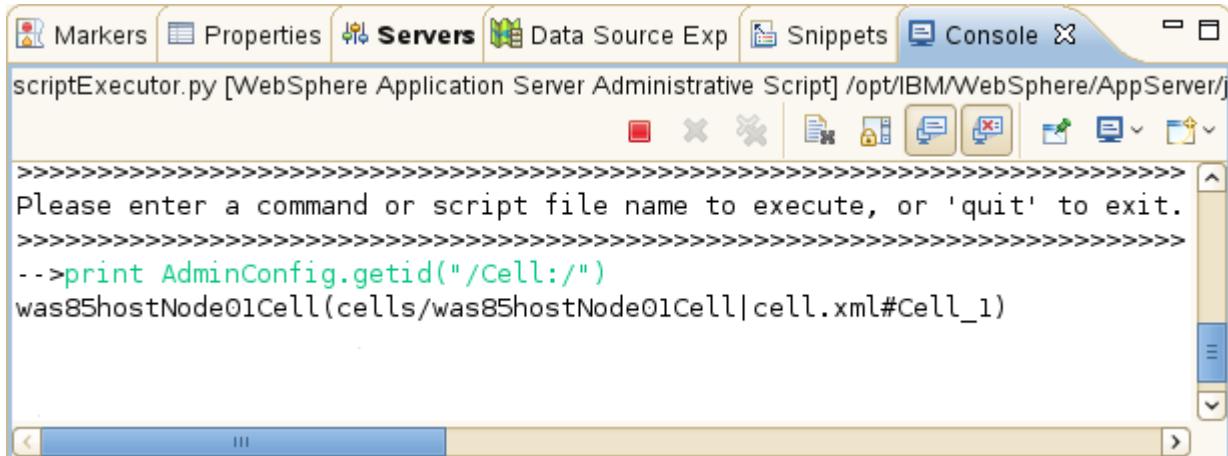
As indicated by the help output, you must supply a containment path as an argument to the method. This path represents the containment hierarchy of the target object and is a string that conforms to the following syntax:

```
"/type1:name1/type2:name2/type3:name3/..."
```

(where `type` is a configuration type name and `name` is the display name of an object).

- \_\_ b. Knowing that you are looking for a **Cell** object, enter the following in the scriptExecutor Console view:

```
print AdminConfig.getId("/Cell:/")
```

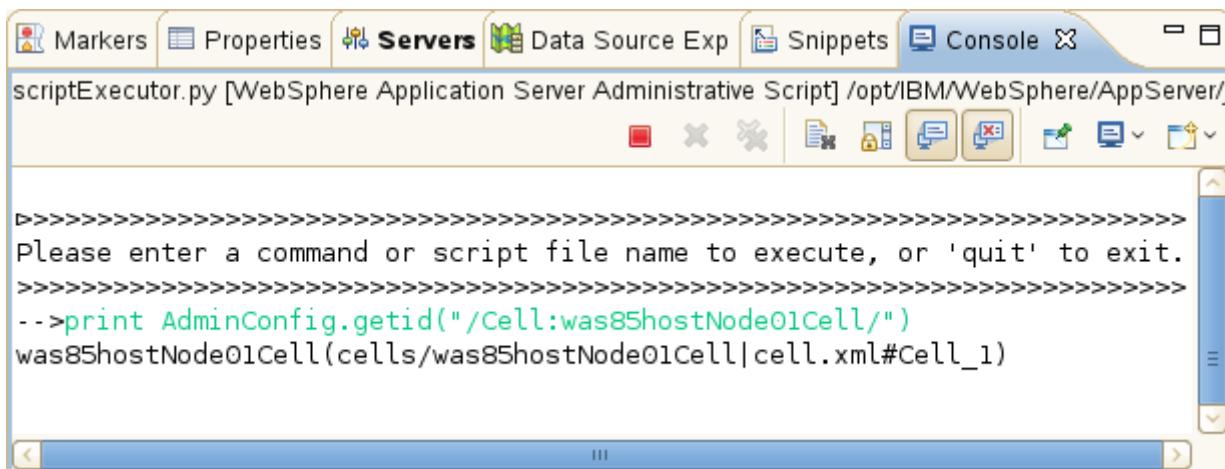


The method returns a string that contains the *configuration name* of the single cell in the configuration. The name consists of two parts: a *display name* (was85hostnode01Cell) and a *configuration ID* (cells/was85hostNode01Cell|cell.xml#Cell\_1). The latter is really what is used to uniquely identify a configuration object.

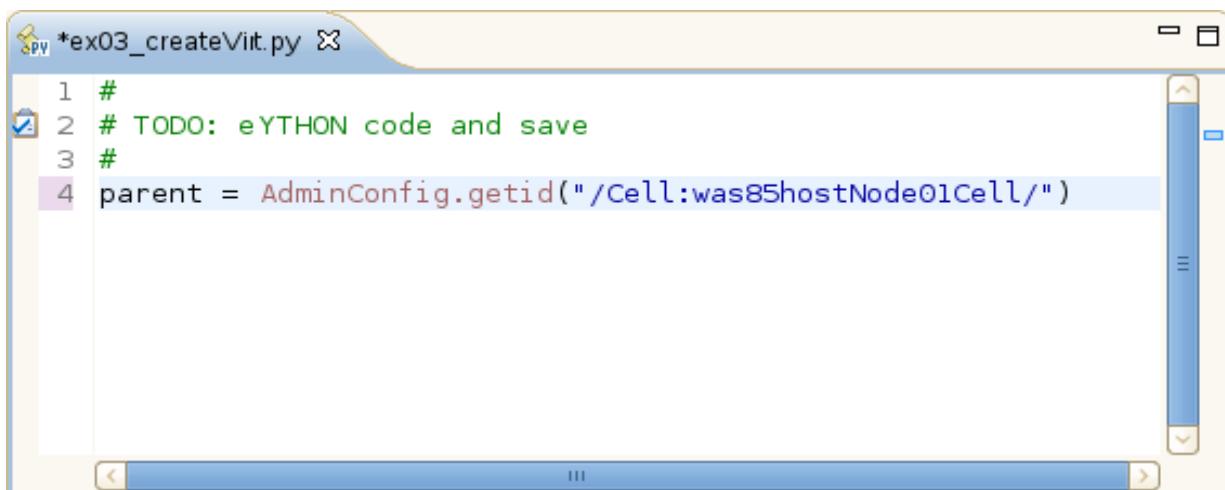
In the method invocation above, the containment path used did not specify the display name of a cell. The method returns more configuration names when there are more cells in the configuration. This "wildcard" form of a containment path is useful when first trying to discover all of the objects in the configuration of a particular type.

- \_\_ c. To narrow down and ensure that you are getting only the configuration name of the `was85hostNode01Cell` cell, specify its display name in the containment path. In the scriptExecutor Console view, enter:

```
print AdminConfig.getId("/Cell:was85hostNode01Cell/")
```

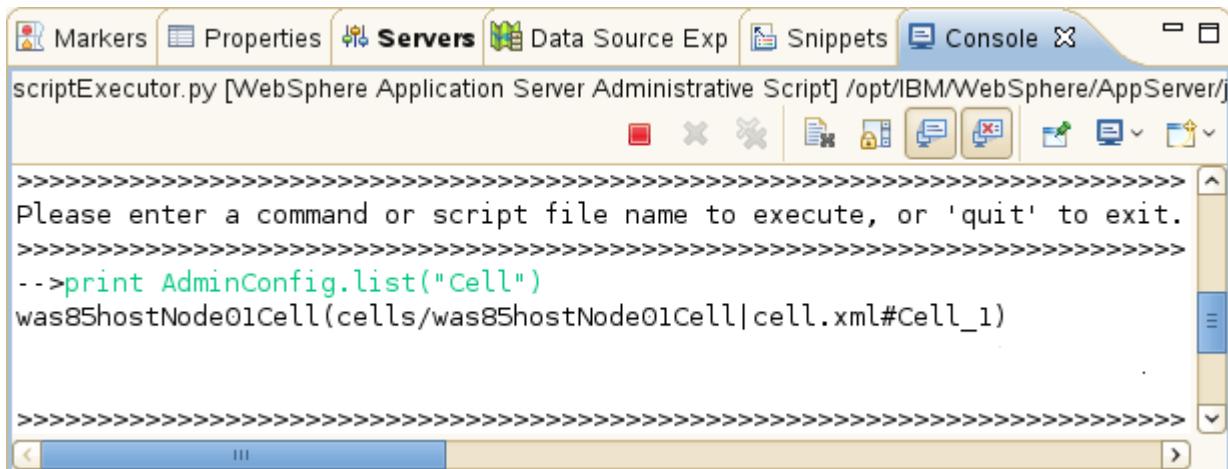


- \_\_ d. Now that you know how to get the configuration ID of cell **was85hostNode01Cell**, assign it to a variable named **parent**, and type the resulting statement in the editor view as shown. You can also cut-and-paste the method invocation from the Console view to the editor view.



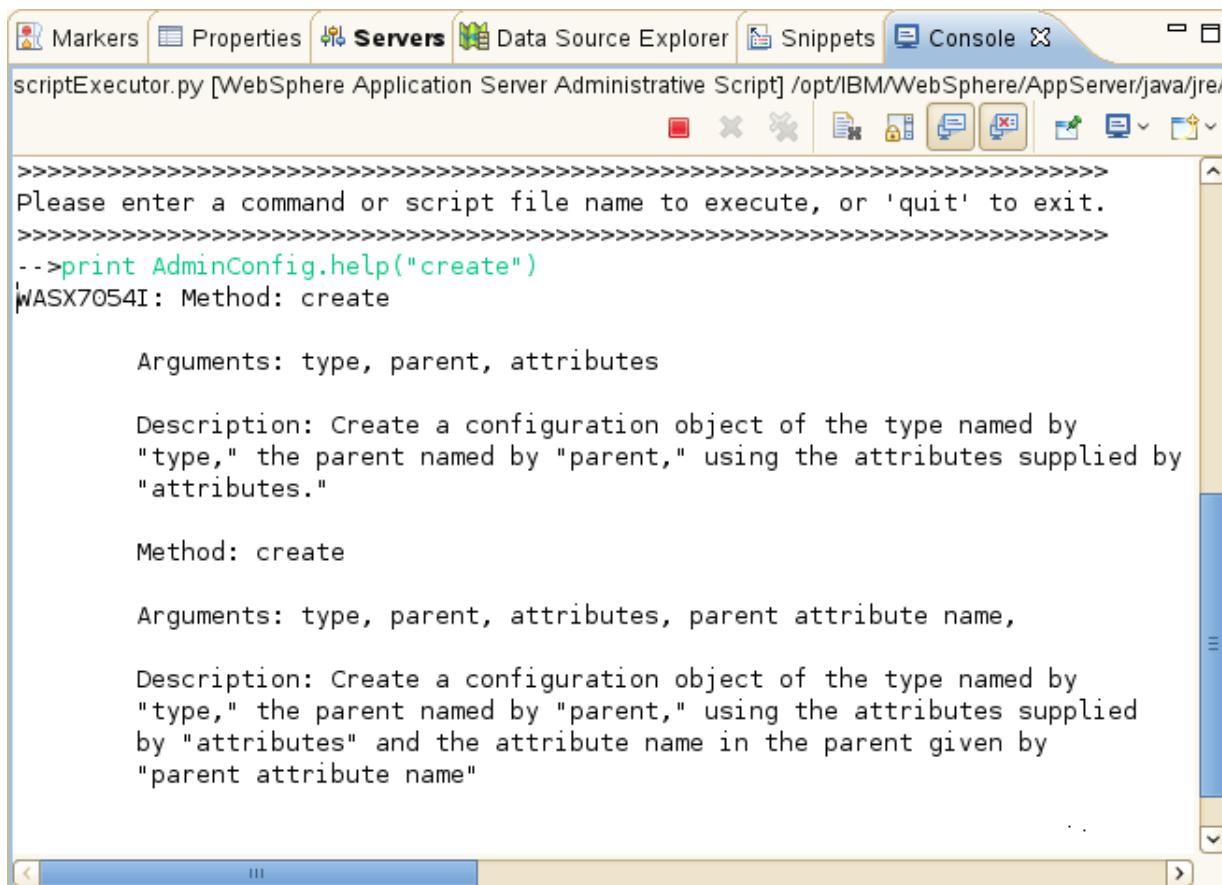
- \_\_\_ e. Save the script file by typing **Ctrl+S**.
  - \_\_\_ f. Another way to retrieve configuration IDs is to use the `list(aConfigurationType)` method. For example, to list the configuration names of all the cells in the configuration, enter:

```
print AdminConfig.list("Cell")
```



- \_\_\_ 4. Next, compose the method to create a new virtual host named **myVirtualHost**.
    - \_\_\_ a. The AdminConfig `create()` method is used to create configuration objects. Start by getting help on its syntax. In the `scriptExecutor` Console view, enter:

```
print AdminConfig.help("create")
```



In its simplest form, the `create()` method takes three arguments:

- `type` is the type of the configuration object to create
- `parent` is the configuration ID for the parent of the object
- `attributes` is a list that contains a name-value pair for each object attribute to be initialized

**Hint**

For complete details on the syntax of all AdminConfig methods, refer to the section titled **Commands for the AdminConfig object using wsadmin scripting** in the WebSphere Application Server V8.5 Information Center.

- \_\_\_ b. You have all the information that is needed to construct the create method. In the editor view, add the following statements after the line you entered earlier:

```
objectType = "VirtualHost"  
attributes = [ ["name", "myVirtualHost"] ]  
AdminConfig.create(objectType, parent, attributes)  
AdminConfig.save()
```

**Information**

Notice in the setting of the `attributes` field, which represents a list of attributes, how the name-value pair for the `name` attribute is itself also expressed as a list. The general format for specifying attributes is:

```
[ [attributeName1, attributeName1], [attributeName2, attributeName2], ... ]
```

Your script looks as follows:

The screenshot shows a Java IDE interface with a Python script open. The title bar reads "ex03\_createVirtualHost.py". The code itself is as follows:

```
1 #  
2 # TODO: enter JYTHON code and save  
3 #  
4 parent = AdminConfig.getId("/Cell:was85hostNode01Cell/")  
5 objectType = "VirtualHost"  
6 attributes = [["name", "myVirtualHost"]]  
7 AdminConfig.create(objectType, parent, attributes)  
8 AdminConfig.save()
```

- \_\_\_ c. Type **Ctrl+S** to save the script.
  - \_\_\_ 5. Before you run the script, display all existing virtual hosts in the configuration so that you can later verify that your script adds one more. In the `scriptExecutor` Console view, enter:

```
print AdminConfig.list("VirtualHost")
```

Two virtual hosts are listed.

6. Run the **ex03\_createVirtualHost** script. In the scriptExecutor Console view, enter:

### ex03 createVirtualHost.py

You might notice the Console view switch to the WebSphere Application Server as the updates to the configuration are written to the console.

- \_\_ 7. Verify that the new virtual host was created. In the scriptExecutor Console view, enter:

```
print AdminConfig.list("VirtualHost")
```

You successfully used the AdminConfig object to create a configuration object.

## **Section 4: Querying and modifying a virtual host object**

Next, create a script to modify the virtual object that you created in the previous step in order to assign values to its `aliases` attribute. In the process, you see the methods used to query configuration object attributes.

Again, referring to Figure 1, the steps to follow are:

1. Get the configuration ID of the object to modify.
  2. Construct and run the appropriate modify method.
  3. Save the configuration change.

— 1. Start by creating a script file named `ex03_modifyVirtualHost.py` in the **Scripts** folder.

  - a. In the Enterprise Explorer view, right-click the **Scripts** folder and select **New > Other**. The Select a wizard dialog opens.
  - b. In the **Select a wizard** dialog, expand **Jython** and select **Jython Script File**. Click **Next**.
  - c. In the **Create a Jython script file** dialog, type `ex03_modifyVirtualHost.py` in the File name field.
  - d. Click **Finish**. The new script file now appears in the Enterprise Explorer view in the **Scripts** folder, and is opened in the Jython editor view.

— 2. The first step is to retrieve the configuration ID of the object to modify. You can use the `getId(aContainmentPath)` method to do so and must supply the containment path of the object.

  - a. What is the containment path for the **myVirtualHost** object? Using the knowledge that you gained from the previous section, try to figure it out. Construct the appropriate `getId()` method invocation and test it in the `scriptExecutor` Console view.

If you get stuck, the solution is shown.

- \_\_ b. In the editor view for **ex03\_modifyVirtualHost.py**, type a statement that retrieves the configuration ID of myVirtualHost and assigns it to a variable named **myVhID**. Your script should look as follows:

```

1 #
2 # TODO: enter JYTHON code and save
3 #
4 myVhID = AdminConfig.getId("/Cell:was85hostNode01Cell/VirtualHost:myVirtualHost/")

```



### Information

To display line numbers in the Jython editor, right-click anywhere in the editor view and select **Preferences**. In the Preferences window, select the **Show line numbers** check box and click **OK**.

- \_\_ c. Type **Ctrl+S** to save the script.
- \_\_ 3. Remember that when you created **myVirtualHost**, you supplied a value for its `name` attribute and not for `aliases`. Confirm that by querying the object by using the `showAttribute()` and `show()` methods. In the editor view, add the following lines to the script:

```

print AdminConfig.showAttribute(myVhID, "name")
print AdminConfig.show(myVhID, ["name", "aliases"])

```

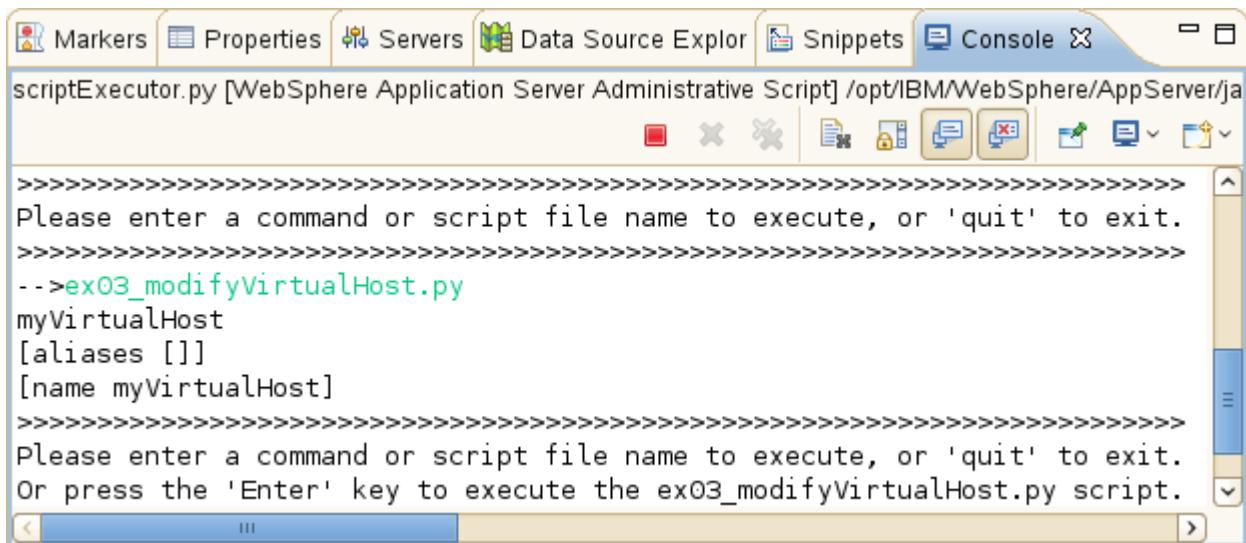
Your script should look as follows:

```

1 #
2 # TODO: enter JYTHON code and save
3 #
4 myVhID = AdminConfig.getId("/Cell:was85hostNode01Cell/VirtualHost:myVirtualHost/")
5 print AdminConfig.showAttribute(myVhID, "name")
6 print AdminConfig.show(myVhID, ["name", "aliases"])
7

```

- \_\_ d. Type **Ctrl+S** to save the script.
- \_\_ e. Run the script: In the scriptExecutor window, enter  
`ex03_modifyVirtualHost.py`



```

Markers Properties Servers Data Source Explor Snippets Console
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/ja
Please enter a command or script file name to execute, or 'quit' to exit.
-->ex03_modifyVirtualHost.py
myVirtualHost
[aliases []]
[name myVirtualHost]
Please enter a command or script file name to execute, or 'quit' to exit.
Or press the 'Enter' key to execute the ex03_modifyVirtualHost.py script.

```

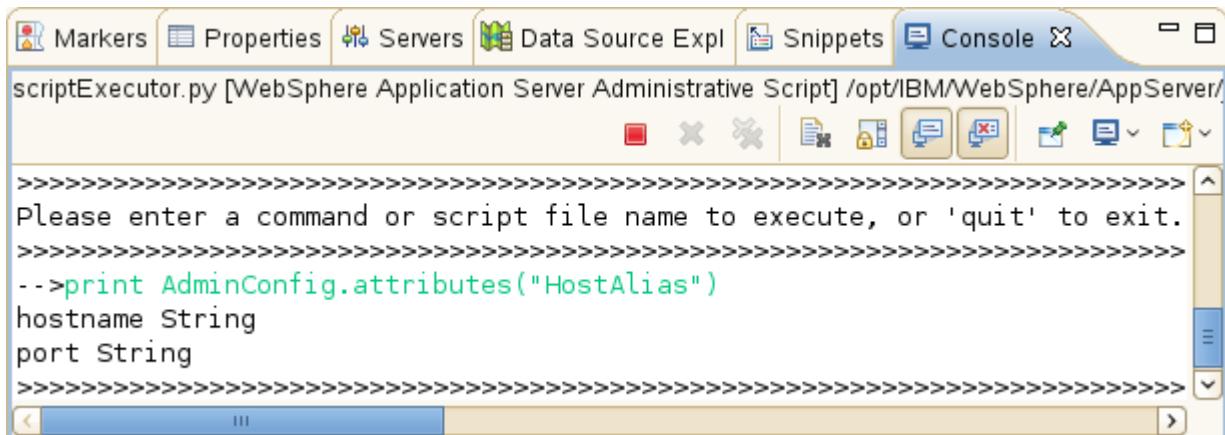
Observe that the `aliases` attribute is an empty collection.

- \_\_\_ 4. You are now ready to construct the `modify` method to assign a value to the `aliases` attribute of `myVirtualHost`.
- \_\_\_ a. First, recall from the previous section that the `aliases` attribute represents a collection of objects of type `HostAlias`. When you explored it, you found that the attributes of the `HostAlias` type are

```

hostName String
port String

```



```

Markers Properties Servers Data Source Explor Snippets Console
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/ja
Please enter a command or script file name to execute, or 'quit' to exit.
-->print AdminConfig.attributes("HostAlias")
hostname String
port String

```

Therefore, in to "create" a `HostAlias` object, you need supply values for its host name and port attributes.

- \_\_\_ b. In the script, set the `port` and `hostName` attributes of a host alias object to 80 for `port` and `host1.ibm.com` for host name, and assign the list to a variable named `hostAlias`. In the editor view, delete the two print statements and type the following line:

```
hostAlias = [{"port": "80"}, {"hostName": "host1.ibm.com"}]
```

Notice how each attribute is set by using a list that contains name-value pairs. All attribute settings are then wrapped in a containing list.

- \_\_\_ c. Compose the method to modify the `aliases` attribute of the virtual host object. In the editor view, add the following lines:

```
AdminConfig.modify(myVhID, [[ "aliases", [hostAlias]]])
```

Note how the value of the `aliases` attribute is expressed as a list because it is a collection.

- \_\_\_ d. Add a line to start the `showall()` method to recursively display the value of all of the attributes of **MyVirtualHost**. This line allows you to verify the results of the modification.

```
print AdminConfig.showall(myVhID)
```

- \_\_\_ e. Finally, add a line to save your configuration change:

```
AdminConfig.save()
```

Your script should look as follows:

```
*ex03_modifyVirtualHost.py ✘
1 #
2 # TODO: enter JYTHON code and save
3 #
4 myVhID = AdminConfig.getid("/cell:was85hostNode01Cell/VirtualHost:myVirtualHost/")
5 hostAlias = [["port", "80"], ["hostname", "host1.ibm.com"]]
6 AdminConfig.modify(myVhID, [[ "aliases", [hostAlias]]])
7 print AdminConfig.showall(myVhID)
8 AdminConfig.save()
```

- \_\_\_ f. Type **Ctrl+S** to save the script.

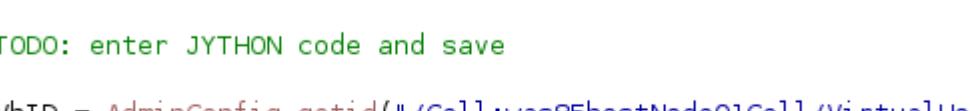
- \_\_ g. Run the script: In the scriptExecutor window, press **Enter** (the scriptExecutor remembers the last script that was run and then runs it),

The `aliases` attribute now has a host alias that is assigned to it and `port` has a port number that is assigned to it.

## Information

Although you did not set the `mimeTypes` attribute when you created `myVirtualHost`, it has values that are assigned to it. This result is because when you create a virtual host, a default is set of 90 MIME entries are automatically assigned to it.

- \_\_\_ 5. What happens if you modify the aliases collection again?
    - \_\_\_ a. Edit the script to specify a value of "host2.ibm.com" for the hostname attribute.



```
1 #  
2 # TODO: enter JYTHON code and save  
3 #  
4 myVhID = AdminConfig.getId("/Cell:was85hostNode01Cell/VirtualHost:m  
5 hostAlias = [["port", "80"], ["hostname", "host2.ibm.com"]]  
6 AdminConfig.modify(myVhID, [[["aliases", [hostAlias]]]])  
7 print AdminConfig.showall(myVhID)  
8 AdminConfig.save()
```

- b. Type **Ctrl+S** to save the script.

- \_\_ c. Run the script: In the scriptExecutor window, press Enter.,

```
Please enter a command or script file name to execute, or 'quit' to exit.
Or press the 'Enter' key to execute the ex03_modifyVirtualHost.py script.
-->
[aliases [[[hostname host1.ibm.com]
[port 80]] [[hostname host2.ibm.com]
[port 80]]]]
[mimeTypes [[[extensions STEP;STP;step;stp]
[type application/STEP]] [[extensions DGW;dgw]
[type application/acad]] [[extensions CCAD]
[type application/clariscad]] [[extensions DRW]]]
```

The new host alias, "host2.ibm.com", is **added** to the collection. The default behavior when modifying a collection is to add the specified entry. To replace the entire collection, change it to an empty list first ([]), and then append the replacement entry.

- \_\_ 6. As an extra check, use the administrative console to verify that the **myVirtualHost** object was properly modified.
- \_\_ a. Display the Servers view by clicking the **Servers** tab.
  - \_\_ a. Right-click **WebSphere Application Server v8.5 at localhost**, and select **Run administrative console**.
  - \_\_ b. Click **OK** at the Invalid Certificate window and proceed.
  - \_\_ c. The administrative console opens in the Admin Console view and displays the Login page. Type `wasadmin` for **User ID**, `web1sphere` for **Password** and click **Log in**.

- \_\_\_ d. The main page of the administrative console is displayed. In the navigation pane, select **Environment > Virtual Hosts**. The virtual hosts that are defined in the configuration are shown in the right pane.

**Virtual Hosts**

**Virtual Hosts**

[+] Preferences

New... Delete

Select Name ▾

You can administer the following resources:

<input type="checkbox"/>	<a href="#">admin_host</a>
<input type="checkbox"/>	<a href="#">default_host</a>
<input type="checkbox"/>	<a href="#">myVirtualHost</a>

Total 3

- \_\_\_ e. In the right pane, select **myVirtualHost**. The name of the virtual host is displayed under General Properties.

**Virtual Hosts > myVirtualHost**

Configuration

**General Properties**

\* Name   

Additional Properties

- [Host Aliases](#)
- [MIME Types](#)

Apply OK Reset Cancel

- \_\_\_ f. Under the Additional Properties, select **Host Aliases**. The host aliases that you defined for **myVirtualHost** are displayed.

The screenshot shows the 'Virtual Hosts' administrative interface. In the center, under 'Host Aliases', there is a table listing two host aliases: 'host1.ibm.com' and 'host2.ibm.com'. Both entries are highlighted with a red rectangular border. The table has columns for 'Select' (checkbox), 'Host Name' (link), and 'Port' (value '80'). At the bottom of the table, it says 'Total 2'.

Select	Host Name	Port
<input type="checkbox"/>	<a href="#">host1.ibm.com</a>	80
<input type="checkbox"/>	<a href="#">host2.ibm.com</a>	80

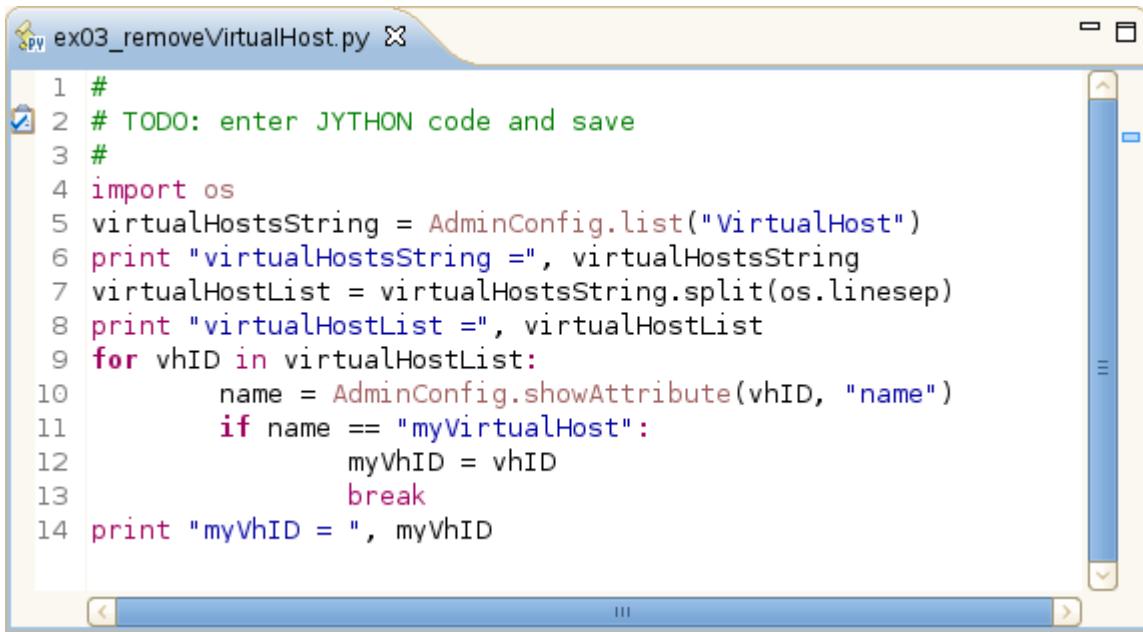
- \_\_\_ g. Click **Logout** to exit the administrative console and close the Admin Console view.

You successfully used the AdminConfig object to modify a configuration object.

## Section 5: Removing a virtual host object

Finally, create a script to remove the virtual object that you created and modified in the previous sections. The steps that are involved are similar to the create and modify changes and, as depicted in Figure 1, consist of:

1. Get the configuration ID of object to remove.
  2. Construct and run the appropriate remove method.
  3. Save the configuration change.
- 1. Start by creating another script file named `ex03_removeVirtualHost.py` in the **Scripts** folder.
- 2. The first step is to retrieve the configuration ID of the object to remove. So far, you used the `getId(aContainmentPath)` method to do so. This method requires knowing the containment path of the object. However, there are many situations where you do not immediately know the containment path of an object. Instead, you know its configuration type and the value of one of its attributes. In such a case, use the `list(aConfigurationType)` method to get a string that contains a list of all the configuration names of objects of this type. You can then search the returned list for the specific object that has the known attribute value.
- a. For the **myVirtualHost** object, you know that its type is **VirtualHost**, and the value of its `name` attribute is "myVirtualHost". In the editor view for `ex03_removeVirtualHost.py`, type the statements that are shown:



```

1 #
2 # TODO: enter JYTHON code and save
3 #
4 import os
5 virtualHostsString = AdminConfig.list("VirtualHost")
6 print "virtualHostsString =", virtualHostsString
7 virtualHostList = virtualHostsString.split(os.linesep)
8 print "virtualHostList =", virtualHostList
9 for vhID in virtualHostList:
10     name = AdminConfig.showAttribute(vhID, "name")
11     if name == "myVirtualHost":
12         myVhID = vhID
13         break
14 print "myVhID = ", myVhID

```

The following is an explanation of the code:

- Line 4 imports the `os` module that contains the definition of the `os.linesep` variable. This variable represents the line separator character and is used in line 7 to split the string that the `list()` method returns.

- Line 5 uses the `list()` method to return a string that contains a list of the configuration names of all objects of type "VirtualHost", and assigns it to the `virtualHostsString` variable.
- Line 7 converts `virtualHostsString` into a list by using the `split()` method with the `os.linesep` character as its parameter, and assigns it to the `virtualHostList` variable.
- Line 9 starts a `for` loop that searches `virtualHostList` for configuration name of the **myVirtualHost** object.
- Line 10 uses the `showAttribute()` method to get the value of the current virtual host object's `name` property.
- Lines 11 - 13 test if the current virtual host's `name` attribute value is "**myVirtualHost**". If so, the correct virtual host object is found and its configuration name is assigned to the `myVhID` variable.

\_\_\_ b. Run the script: In the scriptExecutor Console window, enter:

**ex03\_removeVirtualHost.py**

```

Markers Properties Servers Data Source Explorer Snippets Console >
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/java/jre/bin/java (Jan 25, 2013 9:45:45 AM)
Please enter a command or script file name to execute, or 'quit' to exit.
-->ex03_removeVirtualHost.py
virtualHostsString = admin_host(cells/was85hostNode01Cell|virtualhosts.xml#VirtualHost_2)
default_host(cells/was85hostNode01Cell|virtualhosts.xml#VirtualHost_1)
myVirtualHost(cells/was85hostNode01Cell|virtualhosts.xml#VirtualHost_13591576066663)
virtualHostList = ['admin_host(cells/was85hostNode01Cell|virtualhosts.xml#VirtualHost_2)', 'def
myVhID = myVirtualHost(cells/was85hostNode01Cell|virtualhosts.xml#VirtualHost_13591576066663)

```

The configuration ID of **myVirtualHost** is correctly retrieved.

\_\_\_ 3. Next, compose the method to delete the virtual host object. In the editor view:

- Comment out the print statements (lines 6, 8 and 14)
- Add the following lines:

```

AdminConfig.remove (myVhID)

AdminConfig.save()

print AdminConfig.list("VirtualHost")

```

Your script should look as follows:::

```
spy *ex03_removeVirtualHost.py <input>
1  #
2  # TODO: enter JYTHON code and save
3  #
4  import os
5  virtualHostsString = AdminConfig.list("VirtualHost")
6  #print "virtualHostsString =", virtualHostsString
7  virtualHostList = virtualHostsString.split(os.linesep)
8  #print "virtualHostList =", virtualHostList
9  for vhID in virtualHostList:
10      name = AdminConfig.showAttribute(vhID, "name")
11      if name == "myVirtualHost":
12          myVhID = vhID
13          break
14  #print "myVhID = ", myVhID
15  AdminConfig.remove(myVhID)
16  AdminConfig.save()
17  print AdminConfig.list("VirtualHost")
```

- \_\_\_ c. Type **Ctrl+S** to save the script.
  - \_\_\_ 4. Run the script. In the scriptExecutor Console window, press Enter.

The **myVirtualHost** object no longer appears in the list of virtual host objects. You used the AdminConfig object to delete a configuration object.

## **Section 6: Cleaning up the environment**

- \_\_\_ 1. Stop the scriptExecutor script. In the scriptExecutor Console window, enter: `quit`.
- \_\_\_ 2. Close any scripts that are still in the editor.
- \_\_\_ 3. Stop the server.
  - \_\_\_ a. Display the Servers view by clicking the **Servers** tab.
  - \_\_\_ b. Make sure the **WebSphere Application Server v8.5 at localhost** server is selected and click **Stop the server**.



Wait until the Servers view displays a status of "Stopped".

- \_\_\_ 4. Exit the IBM Assembly and Deploy Tools by selecting **File > Exit**.

**End of exercise**

## Exercise review and wrap-up

In this exercise, you used the AdminConfig object to create, modify, query, and delete a virtual host configuration object. You followed the steps for making a configuration change with the AdminConfig object and identified the primary methods and resources that are required to successfully complete each step. You developed and ran the resulting Jython scripts in the IBM Assembly and Deploy Tools.



# Exercise 4. Using the AdminApp object

## What this exercise is about

In this exercise, you learn how to use the AdminApp administrative object to query, install, update, and uninstall an application. By developing scripts to complete these application management tasks, you gain familiarity with the primary methods and resources that are required to effectively use the AdminApp object.

## What you should be able to do

At the end of this exercise, you should be able to:

- Access online help about how to use the AdminApp object
- Employ the primary methods and resources that are required to use the AdminApp object effectively
- Use the AdminApp object to query, install, edit, update, and uninstall an application

## Introduction

The AdminApp administrative object is used to manage applications and supports a comprehensive set of application management functions. In this exercise, you use the AdminApp object to perform the following application management tasks:

- Query the list of installed applications
- Install an application with basic options
- Uninstall an application
- Install an application with task options
- Update a module inside an application

The application that you install and manage is supplied in an Enterprise Archive (EAR) file named `myIVT.ear`. It is a version of the Installation Verification Test (IVT) application that comes with WebSphere Application Server. Because it requires no database resources to connect to, it is a simple application to use in exploring the capabilities of the AdminApp object.

The different AdminApp methods to use for each supported application management task is shown in Figure 1.

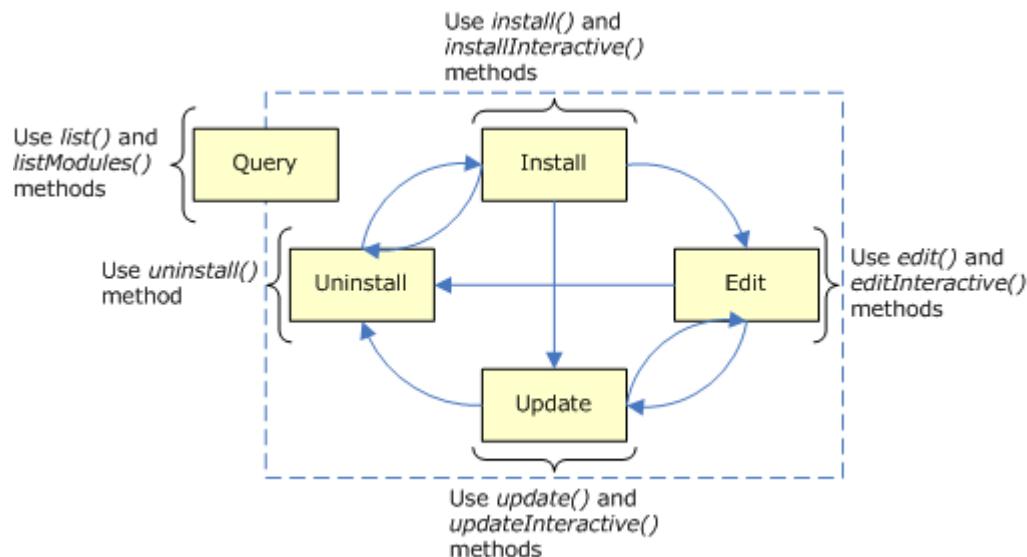


Figure 1 - Application management tasks that the AdminApp object supports

Using the IBM Assemble and Deploy Tools (IADT), you develop Jython scripts that use these primary methods of the AdminApp object.

# Requirements

To complete this exercise, you need the WebSphere Application Server Network Deployment V8.5 product image that is installed and the stand-alone *server1* application server that is created in *SamplesProfile*. In addition, the IBM Assembly and Deploy Tools for WebSphere Administration 8.5 (IADT) should already be installed.

## Exercise instructions



### Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

`<was_root>`: /opt/IBM/WebSphere/AppServer

`<profile_root>`: /opt/IBM/WebSphere/AppServer/profiles

### Section 1: Preparing the environment

In this step, you open the same IBM Assembly and Deploy Tools (IADT) workspace that you used in the previous exercise, namely, *StandAloneWS*. This workspace already contains a Jython project definition and the definition of the *SamplesProfile server1* stand-alone server. It also already includes the utility script, *scriptExecutor.py*, which is used to facilitate script execution inside the IADT.



### Note

If you must create or re-create the StandAloneWS workspace from scratch, follow the steps that are described in **Section: 1 Preparing the environment of Exercise 2**.

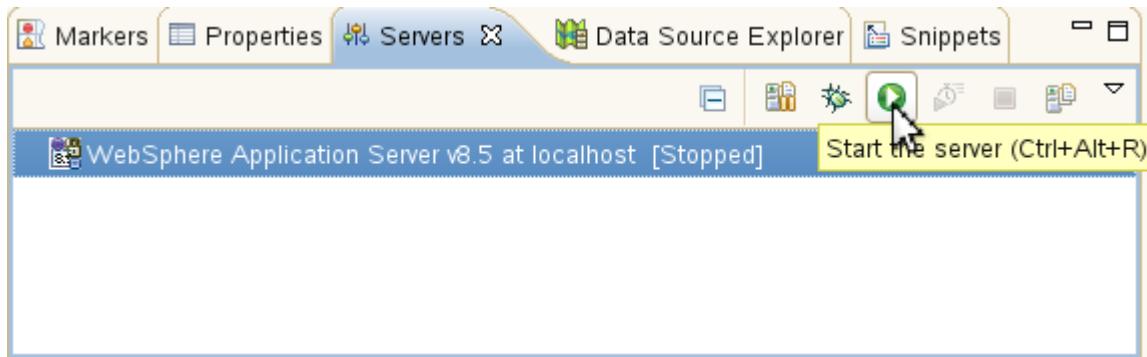
- \_\_\_ 1. Start the IBM Assembly and Deploy Tools and open the **StandAloneWS** workspace.
  - \_\_\_ a. On the desktop, double-click the **IBM Assembly and Deploy Tools** icon.
  - \_\_\_ b. In the Workspace Launcher window, make sure that the Workspace field has a value of **/usr/LabWork/StandAloneWS**.
  - \_\_\_ c. Click **OK**. The IADT opens and displays the Java EE perspective.
- \_\_\_ 2. Start the **SamplesProfile server1** server.



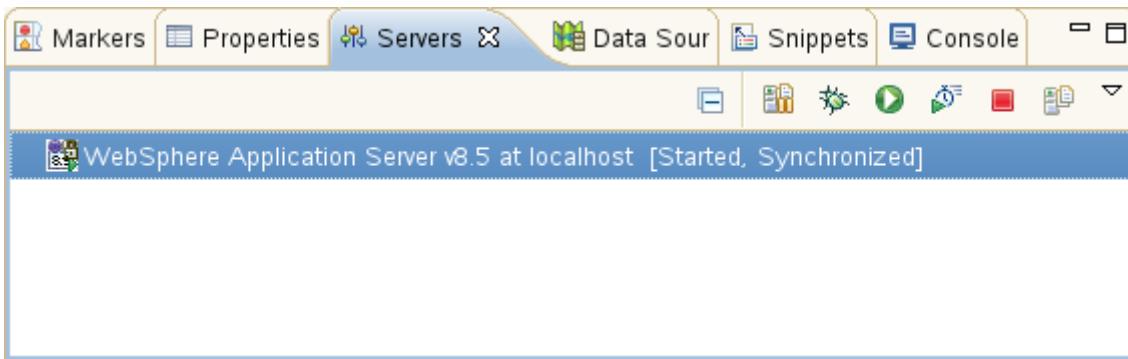
## Information

Using the AdminApp object does not require a connection to a running server. Therefore, the scripts that are developed in this lab can run without being connected to *SamplesProfile server1*. However, because you open the administrative console later in the exercise to verify your configuration changes, you are asked to start the server now for convenience.

- \_\_ a. In the Servers view, make sure that **WebSphere Application v8.5 Server at localhost** is selected, and click the **Start the server** button.

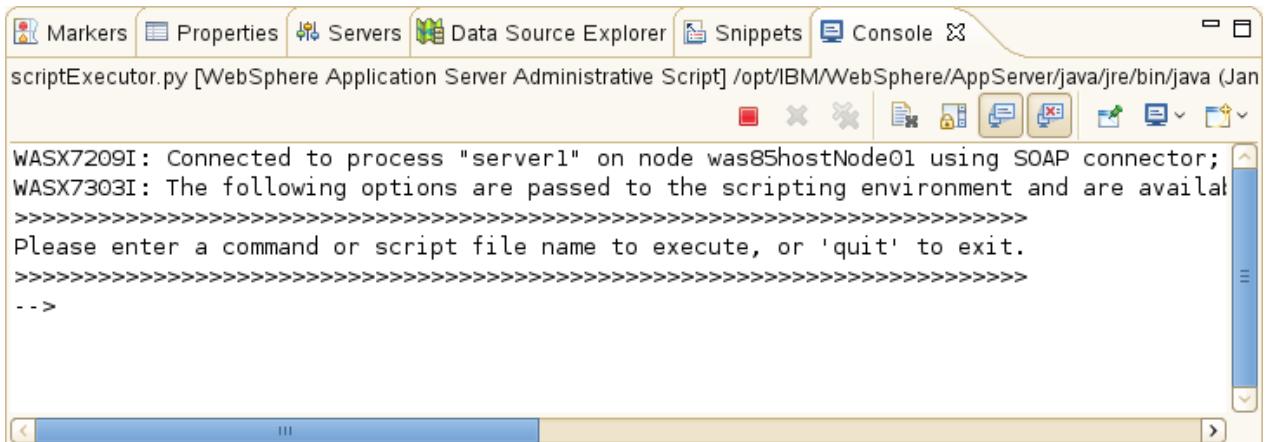


- \_\_ b. Wait until the server is started. As the server starts, focus is switched to the **Console** view where startup messages are displayed. When the server is started, the Servers view displays a status of **Started**.



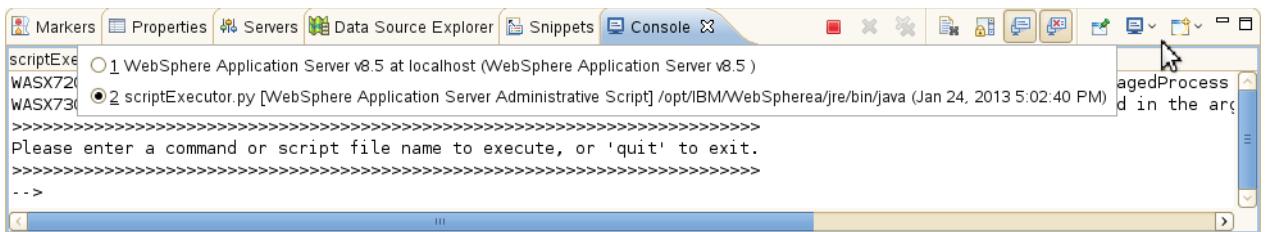
- \_\_ 3. Run the scriptExecutor utility script.
  - \_\_ a. In the Enterprise Explorer view, expand **AdminScriptingProject > Utilities**. Right-click **scriptExecutor.py** and select **Run As > Administrative Script**.
  - \_\_ b. After a few moments, the Console view displays a message that confirms the establishment of a wsadmin connection with the server1 server. An input prompt

is also displayed for running an administrative command or a script file. You are now ready to develop and run your scripts.



### Information

Recall that you have two Console views opened: one for the *scriptExecutor* script and the other of the *server1* standard output. To switch between Console views, click the arrow of the **Display Selected Console** button and select the console output to view.



## Section 2: Querying the existing applications and modules

Start by exploring the AdminApp methods that allow you to list the applications and application modules that are installed in your environment.

- \_\_\_ 1. What applications are installed in the environment? You can use the `list()` method to find out.
- \_\_\_ a. In the scriptExecutor Console view, enter:

```
print AdminApp.list()
```



### Information

When you gain focus on the scriptExecutor Console view, the cursor is not immediately positioned on the input prompt line. However, you can start typing a command and the cursor is automatically repositioned on the input prompt line.

```
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/j
Please enter a command or script file name to execute, or 'quit' to exit.
-->print AdminApp.list()
DefaultApplication
IBMUTC
ivtApp
query
```

The display names of all installed applications are returned.

- \_\_\_ b. You can also specify a scope as a parameter of the `list()` method. In the scriptExecutor Console view, enter the following command on one line:

```
print
AdminApp.list("WebSphere:cell=was85hostNode01Cell,node=was85hostNode0
1,server=server1")
```

Since the environment is stand-alone, the scope parameter does not make a difference, and the same list of application names as before is displayed.

- \_\_ 2. Next, use the `listModules()` method to determine the list of modules that are contained in the `DefaultApplication` application.

- a. In the scriptExecutor Console view, enter:

```
print AdminApp.listModules("DefaultApplication")
```

The DefaultApplication application contains two modules: the DefaultApplication web module and the Increment EJB module.

- \_\_ b. Additionally, the `listModules()` method can show to which server the modules are mapped. In the `scriptExecutor` Console view, enter:

```
print AdminApp.listModules("DefaultApplication", "-server")
```

The name of the server to which a module is mapped is displayed after the module name.

You explored the application querying capabilities of the AdminApp object.

### Section 3: Performing a basic application installation

In this section, you use the `install()` method of the `AdminApp` object to install the MyIVT application with minimal options. You start by using the `help()` method to discover the correct invocation syntax and parameter list. You then use `options()` method to discover the options available for installing the application. Finally, you compose the method, run it, and verify the results by using the administrative console.

- \_\_\_ 1. First, create a script file named `ex04_installApp.py` in the **Scripts** folder.
  - \_\_\_ a. In the Enterprise Explorer view, right-click the **Scripts** folder and select **New > Other**. The Select a wizard dialog opens.
  - \_\_\_ b. In the **Select a wizard** dialog, expand **Jython** and select **Jython Script File**. Click **Next**.
  - \_\_\_ c. In the **Create a Jython script file** dialog, type `ex04_installApp.py` in the File name field.
  - \_\_\_ d. Click **Finish**. The new script file is now shown in the Enterprise Explorer view in the Scripts folder, and is opened in the Jython editor view.
- \_\_\_ 2. Now begin by getting help on how to use the `install()` method. In the `scriptExecutor` Console view, enter:

```
print AdminApp.help("install")
```

```
Markers Properties Servers Data Source Explorer Snippets Console < >
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/java.
Please enter a command or script file name to execute, or 'quit' to exit.
-->print AdminApp.help("install")
WASX7096I: Method: install

Arguments: filename, options

Description: Installs the application in the file specified by "filename" using the options specified by "options". All required information must be supplied in the options string; no prompting is performed.

The AdminApp "options" command may be used to get a list of all possible options for a given ear file. The AdminApp "help" command may be used to get more information about each particular option.
```

The help for the `install()` method indicates that it takes two arguments:

- The **filename** of the application to install

- An **options** string (or list) specifying the options that are used for the installation

It further indicates that the `options()` method can be used to determine the list of valid options for an EAR file.



## Information

Options identify the targets of the method and specify what tasks it performs. Specifically, they can be "switches" (for example, `"-nopreCompileJSPs"`), single-valued options (for example, `["-server", aServerName]`), or task options (for example, `["MapWebModtoVH", taskOptionValues]`). Options vary depending on the method and the contents of the application.

3. Determine the list of options available for installing the MyIVT application.

\_\_ a. In the scriptExecutor Console view, enter:

```
print AdminApp.options ("/usr/Software/Ears/MyIVT.ear")
```

The `options()` method returns the list of options available for installing the `MyIVT.ear` file. It derives this list by looking at the deployment descriptors of the EAR file and identifying the options that might apply. Although the list represents only the options that can be specified for the application, it is still relatively long. However, it is not necessary to specify all of them. The values for most of them are more than likely already defined in the EAR file deployment descriptors. When installing an application, if options are omitted, default bindings are generated and used.



## Information

Invoking the `options()` method without any parameters returns the list of options applicable to all EAR files.

- \_\_\_ b. In the Console view, scroll down to the end of the list. Two options of particular interest are `node` (which specifies the node where the application is installed) and `server` (which specifies the server where the application is installed). These are the installation targets..

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Contains tabs for "Markers", "Properties", "Servers", "Data Source Explorers", "Snippets", and "Console". The "Console" tab is selected.
- Title Bar:** Displays the file path "scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/ja...
- Toolbar:** Includes icons for red square (close), grey X (minimize), and other system functions.
- Content Area:** Shows the command-line interface of the administrative script. It lists command options: target, server, cell, cluster, contextroot, and custom. Below these, it prompts the user to enter a command or script file name to execute, or 'quit' to exit. A command "- ->" is shown at the bottom.

- \_\_ c. You can also get help on the different options that the AdminApp methods use. For example, to learn more about the `server` and `node` options, in the `scriptExecutor` Console view, enter following lines, one at a time:

```
print AdminApp.help("node")  
print AdminApp.help("server")
```

A description of each method option is displayed. They indicate that both options are single-valued; therefore, they can be specified in an options list by using the following format:

```
["-node", aNodeName, "-server", aServerName]
```

- \_\_\_ 4. You are now ready to compose an `install()` method to install `/usr/Software/Ears/MyIVT.ear` on `server1` of node `was85hostNode01`. Try to figure out the correct method invocation syntax, and type it in the editor view of `ex04_installApp.py`. Also, remember to include a statement to save your configuration change.



### Hint

Remember that you can also use the code assist feature of the Jython editor to get help about method signatures (type **Ctrl+Space** as you compose the method to start it).

You can peek at the screen capture that is shown if you are stuck.

The screenshot shows a Jython editor window titled "ex04\_installApp.py". The code in the editor is as follows:

```
1 #
2 # TODO: enter JYTHON code and save
3 #
4 AdminApp.install("/usr/Software/Ears/MyIVT.ear", ["-node", "was85hostNode01", "-server", "server1"])
5 AdminConfig.save()
```

- \_\_\_ 5. Type **Ctrl+S** to save the script.  
\_\_\_ 6. Run the `ex04_installApp.py` script. In the scriptExecutor console view, enter:  
`ex04_installApp.py`

As the installation progresses, installation messages are displayed in the Console. When finished, a message that states that the application is installed successfully is shown.



## Note

During installing, the console display might switch to the server1 standard output view. To switch back to the scriptExecutor view, click **Display Select Console**.

- \_\_\_ 7. Using the administrative console, verify that the application was indeed successfully installed.

\_\_\_ a. Display the Servers view by clicking the **Servers** tab.

\_\_\_ a. Right-click **WebSphere Application Server v8.5 at localhost**, and select **Run administrative console**.

\_\_\_ b. In the Security Alert window, click **OK** to accept the security certificate and proceed.

\_\_\_ c. The administrative console opens in the Admin Console view and displays the Login page. Type `wasadmin` for **User ID**, `websphere` for **password**, and click **Log in**.

- \_\_\_ d. The main page of the Administrative console is displayed. In the navigation pane, select **Applications > Application Types > WebSphere enterprise applications**.

The screenshot shows the 'Enterprise Applications' page of the WebSphere Administrative Console. The title bar says 'Enterprise Applications'. Below it, a message says 'Use this page to manage installed applications. A single application can be deployed onto multiple servers.' There's a 'Preferences' link and a toolbar with buttons for Start, Stop, Install, Uninstall, Update, Rollout Update, Remove File, Export, Export DDL, and Export File. Below the toolbar is a toolbar with icons for Select, Name (with a dropdown arrow), and Application Status (with a refresh icon). A section titled 'You can administer the following resources:' lists five applications: DefaultApplication, IBMUTC, My\_IVT\_Application, ivtApp, and query. The 'My\_IVT\_Application' row is highlighted with a red border. At the bottom left, it says 'Total 5'.

Select	Name	Application Status
<input type="checkbox"/>	<a href="#">DefaultApplication</a>	
<input type="checkbox"/>	<a href="#">IBMUTC</a>	
<input type="checkbox"/>	<a href="#">My_IVT_Application</a>	
<input type="checkbox"/>	<a href="#">ivtApp</a>	
<input type="checkbox"/>	<a href="#">query</a>	

The list of applications that are installed on the server is displayed in the right pane and contains the display name of the MyIVT application,  
**My\_IVT\_Application**. Success!

- \_\_\_ e. Since you did not specify any options in the `install()` method, except for the target node and server names, to which virtual host was the web module inside the MyIVT application mapped? To find out, do the following steps:
1. In the Enterprise Applications pane, click **My\_IVT\_Application**.  
The configuration properties of the application are displayed.

2. Under Web Module Properties, click **Virtual hosts**.

Select	Web module	Virtual host
<input type="checkbox"/>	My_IVT_Webapp_Display_Name	default_host

The Virtual hosts pane shows that the **My\_IVT\_Webapp\_Display\_Name** web module is mapped to the **default\_host** virtual host. Make a mental note of this value as you are going to reinstall the application in a subsequent step and map this web module to a different virtual host.

- \_\_\_ f. Click **Logout** to exit the administrative console. Leave the Admin Console view open for later use.



**Hint**

Avoid using the administrative console and wsadmin at the same time to make configuration changes. Because each tool initially applies changes to a local workspace, conflicts can result if both processes are changing the same configuration objects. Even if different objects are affected, sometimes a change that is made in one tool does not immediately reflect in the other. This behavior is why you are asked to log out of the administrative console after using it for verification purposes.

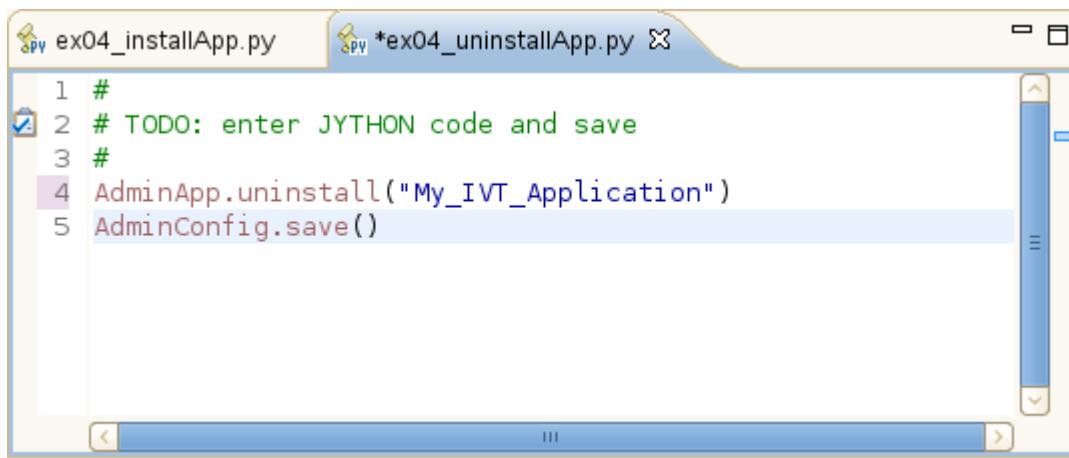
You successfully used the AdminApp object to do a basic application installation.

## Section 4: Uninstalling an application

Uninstall the MyIVT application so that you can reinstall it again with task options in the next step. To do so, use the AdminApp `uninstall()` method.

- \_\_\_ 1. Create another script file named `ex04_uninstallApp.py` in the **Scripts** folder.
- \_\_\_ 2. The `uninstall()` method is simple to use. In its simplest form, it takes only one parameter: the display name of the application to uninstall. In the previous section, you discovered that the display name of the MyIVT application is `My_IVT_Application`. Using this information, compose an `uninstall()` method that uninstalls `My_IVT_Application`, and type it in the editor view of `ex04_installApp.py`. Again, remember to include a statement to save your configuration change.

Your script should look as follows:

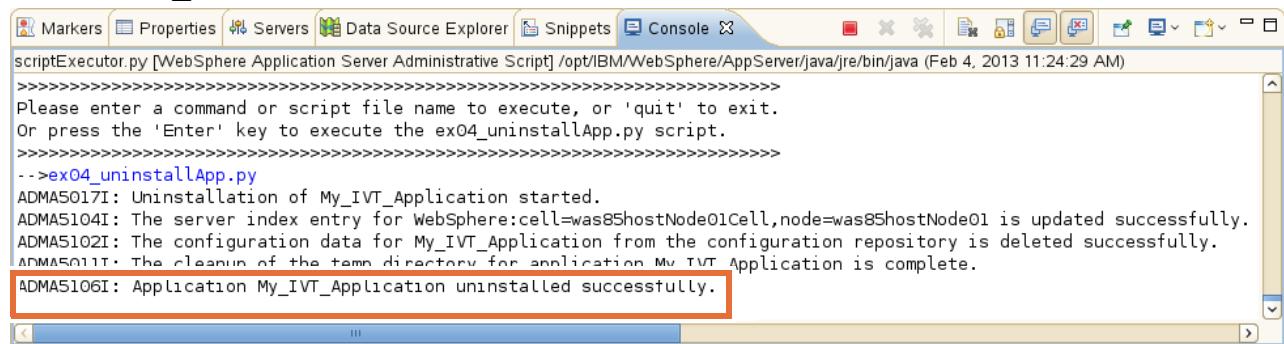


```

 1 #
 2 # TODO: enter JYTHON code and save
 3 #
 4 AdminApp.uninstall("My_IVT_Application")
 5 AdminConfig.save()

```

- \_\_\_ 3. Type **Ctrl+S** to save the script.
- \_\_\_ 4. Run the `ex04_uninstallApp.py` script. In the `scriptExecutor` Console view, enter `ex04_uninstallApp.py`



```

Markers Properties Servers Data Source Explorer Snippets Console >
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/java/jre/bin/java (Feb 4, 2013 11:24:29 AM)
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

The Console view displays a message that the application was successfully uninstalled.

- \_\_\_ 5. Use the administrative console to confirm that the MyIVT is successfully uninstalled.
  - \_\_\_ a. In the Admin Console view, log in using `wasadmin` (for **User ID**) and `websphere` (for **password**).

- \_\_\_ b. In the navigation pane, select **Applications > Application Type > WebSphere enterprise applications**.

The list of applications that are installed on the server no longer contains **My\_IVT\_Application**.

The screenshot shows the 'Enterprise Applications' page in the WebSphere Administrative Console. At the top, there is a toolbar with buttons for Start, Stop, Install, Uninstall, Update, Rollout Update, Remove File, Export, Export DDL, and Export File. Below the toolbar is a section for managing resources, featuring icons for file operations like copy, move, and delete. A search bar allows filtering by 'Name'. A table lists four applications: DefaultApplication, IBMUTC, ivtApp, and query, each with a checkbox and a green circular status icon.

Select	Name	Application Status
<input type="checkbox"/>	<a href="#">DefaultApplication</a>	
<input type="checkbox"/>	<a href="#">IBMUTC</a>	
<input type="checkbox"/>	<a href="#">ivtApp</a>	
<input type="checkbox"/>	<a href="#">query</a>	

Total 4

- \_\_\_ c. Click **Logout** to exit the administrative console. Leave the Admin Console view open for later use.

You successfully used the AdminApp object to uninstall an application.

## ***Section 5: Installing an application with task options***

In an earlier section, you installed the MyIVT application by specifying only the node and server targets as options. In most situations, however, installing an application requires more configuration, such as mapping web modules to specific virtual hosts. You can achieve this configuration by specifying task options in the invocation of the `install()` method. A task represents a step that is performed as part of the execution of the method.

In this section, you use the `install()` method of the `AdminApp` object to install the `MyIVT` application with a task option to map its `My_IVT_Webapp_Display_Name` web module to a virtual host other than `default_host`. You first use the `taskInfo()` method and the WebSphere Application Information Center to discover the appropriate invocation syntax, and then, compose and run the method. Finally, you verify the results by using the administrative console.

- \_\_\_ 1. Begin by identifying the name of the task option to use for mapping a web module to a virtual host. Earlier, you invoked the `options()` method on the `MyIVT.ear` file to get the list of options applicable to the EAR file. Its partial output is shown again.

One of the displayed options is MapWebModToVH. Sounds like the one you are looking for.

- \_\_ 2. Find out more about the `MapWebModToVH` task option for the MyIVT application by using the `taskInfo()` method. In the scriptExecutor Console view, enter:

```
print AdminApp.taskInfo("/usr/Software/Ears/MyIVT.ear",
"MapWebModToVH")
```

The `taskInfo()` method indicates that the `MapWebModToVH` task option specifies three fields. It also shows their default values in the EAR file. By looking at these default bindings, you can derive the following about the three fields:

- `webModule`: The display name of the module to map
  - `uri`: URI that points to the location of the module deployment descriptor. This location is where the virtual host mappings for a web module are kept in the EAR file.
  - `virtualHost`: The name of the virtual host to map to

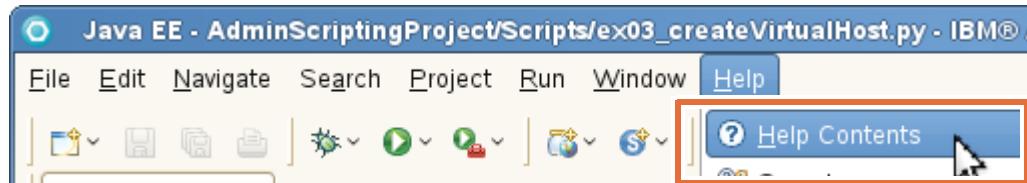
The first two are key fields, and the latter can be assigned a value.



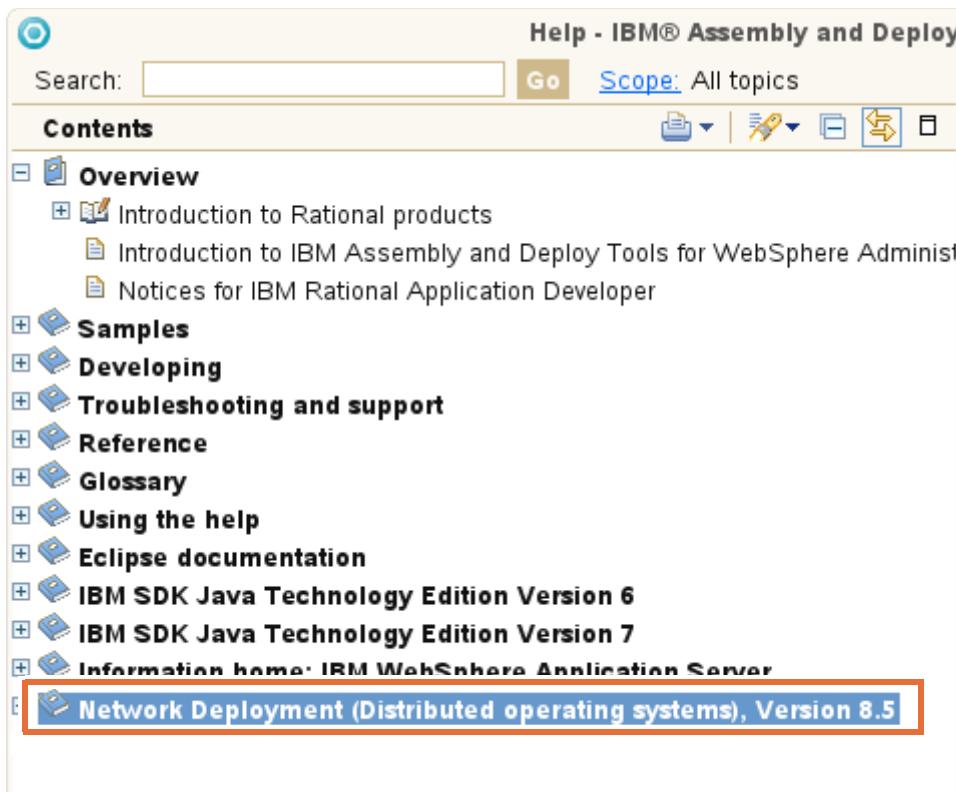
## Information

Task options result in one or more entries or groups of entries (rows) being added to or updated in the configuration data of the application. This result is why an entry has keys (to uniquely identify it) and one or more values that contain actual configuration data.

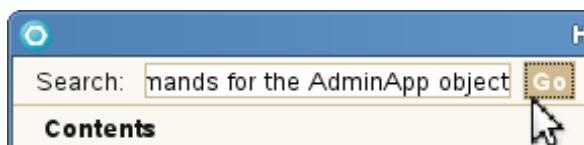
3. You now have an idea of what must be specified for the `MapWebModToVH` task option. But what about the syntax? To find out, go to the WebSphere Application Server Information Center.
- a. On the IADT main menu, click **Help > Help Contents**.



- b. In the Contents view, select **Network Deployment Version 8.5**



- c. In the Search box, type **Commands for the AdminApp object**, and click **GO**. The search returns the list of hits in a Search Results pane.



- \_\_\_ d. In the Search Results pane, click the match named **Commands for the AdminApp object** that is associated with the **Network Deployment (Distributed Operating Systems), Version 8.5.**

The screenshot shows the 'Help - IBM® Assembly' window. In the search bar, 'Commands for the AdminApp object' is entered. The 'Scope' dropdown is set to 'All topics'. The 'Search Results' pane displays a single result titled 'Commands for the AdminApp object using wsadmin scripting'. A red box highlights this title. Below it is a brief description: 'Use the AdminApp object to install, modify, and administer applications.'

- \_\_\_ e. The **Commands for the AdminApp object** page is displayed in the right pane. Scroll down and click the link for **edit**.

The screenshot shows the 'Commands for the AdminApp object' page. It lists various commands available for the AdminApp object. The 'edit' command is highlighted with a red box.

- [deleteUserAndGroupEntries](#)
- **edit**
- [editInteractive](#)
- [export](#)
- [exportDDL](#)
- [exportFile](#)
- [getDeployStatus](#)
- [help](#)
- [install](#)
- [installInteractive](#)
- [isAppReady](#)
- [list](#)
- [listModules](#)
- [options](#)
- [publishWSDL](#)
- [searchJNDIReferences](#)
- [taskInfo](#)
- [uninstall](#)
- [update](#)
- [updateAccessIDs](#)
- [updateInteractive](#)
- [view](#)

- \_\_\_ f. Examples of the syntax of an `edit()` method that uses the **MapWebModToVH** task option are shown. The **Using Jython list** example shows the following format:

- Using Jython list:

```
option = [{"JavaMail 32 Sample WebApp", "mtcomps.war", "WEB-INF/web.xml", "newVH"}]
mapVHOption = [{"-MapWebModToVH", option}]
print AdminApp.edit("JavaMail Sample", mapVHOption)
```

Like other options, you can specify task options by using a Jython list syntax as follows:

```
"-taskName", [optionValue1, optionValue2, ...])
```

- \_\_\_ g. Leave the Information Center browser window open on the **Commands for the AdminApp object** page for later use.
- \_\_\_ 4. Use scripts to create a virtual host. In the previous exercise, Using the AdminConfig object, you developed two scripts to create and modify a virtual host. Run those scripts again by using the scriptExecutor. If you did not complete the previous exercise, the scripts are available in the directory /usr/Solutions/Exercise03.
- \_\_\_ a. In the scriptExecutor Console view, enter:
- ```
ex03_createVirtualHost.py
```
- \_\_\_ b. In the scriptExecutor Console view, enter:
- ```
ex03_modifyVirtualHost.py
```
- \_\_\_ c. Use the Admin Console to verify that the virtual host myVirtualHost was created.
- \_\_\_ 5. You are now ready to compose the `install()` method in the **ex04\_installApp.py** editor view. Modify the script to install MyIVT and map its **My\_IVT\_Webapp\_Display\_Name** module to the `myVirtualHost` virtual host.
- \_\_\_ a. If necessary, open the **ex04\_installApp.py** script in the editor view. In the Enterprise Explorer view, expand **AdminScriptingProject > Scripts** and double-click **ex04\_installApp.py**.
- \_\_\_ b. In the editor view, modify the script as shown.

```

1  #
2  # TODO: enter JYTHON code and save
3  #
4  # EAR file name
5  earFileName = "/usr/Software/Ears/MyIVT.ear"
6  #
7  # Installation target options
8  #
9  node = "was85hostNode01"
10 server = "server1"
11 #
12 # "MapWebModToVH" task options
13 #
14 webModule = "My_IVT_Webapp_Display_Name"
15 uri = ".*"
16 virtualHost = "myVirtualHost"
17 mapWebOptions = [[webModule, uri, virtualHost]]
18 #
19 AdminApp.install(earFileName, ["-node", node, "-server", server, "-MapWebModToVH", mapWebOptions])
20 AdminConfig.save()

```



## Note

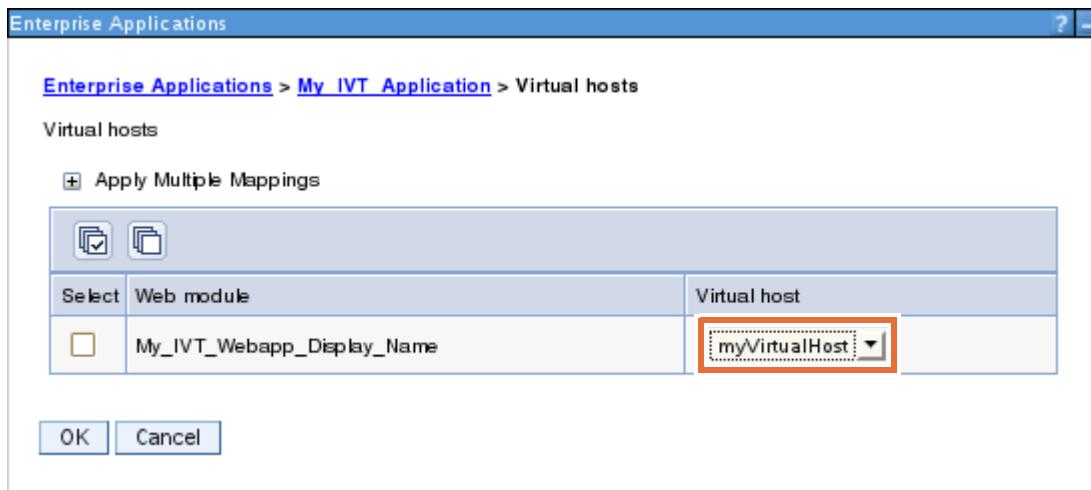
Note the use of variables to break up what would otherwise be a long method invocation line. This scripting style also improves readability and reduces the chance of syntax errors. In addition, it enhances the flexibility of the script, as variables can easily be converted to parameters to make the script more generic.

- \_\_\_ 6. Type **Ctrl+S** to save the script.
  - \_\_\_ 7. Run the ex04\_installApp.py script. In the scriptExecutor Console view, enter:

## ex04\_installApp.py

When execution is finished, a message that states that the application is installed successfully is shown.

- \_\_\_ 8. Use the administrative console to verify that the application was successfully installed, and that the **My\_IVT\_Webapp\_Display\_Name** module was mapped to the `myVirtualHost` virtual host.
- \_\_\_ a. Log in to the administrative console. In the Admin Console view, type `wasadmin` for **User ID**, `web1sphere` for **password**, and click **Log in**. The console main page is displayed.
  - \_\_\_ b. In the navigation pane, select **Applications > Application Type > WebSphere enterprise applications**. The list of applications that are installed on the server is displayed in the right pane.
  - \_\_\_ c. In the Enterprise Applications pane, click **My\_IVT\_Application**. The configuration properties of the application are displayed.
  - \_\_\_ d. Under Web Module Properties, click **Virtual hosts**. The Virtual hosts pane is displayed.



The Virtual hosts pane shows that the **My\_IVT\_Webapp\_Display\_Name** Web module is now mapped to `myVirtualHost`.

- \_\_\_ e. Click **Logout** to exit the administrative console. Leave the Admin Console view open for later use.

You successfully used the `AdminApp` object to install an application installation with task options.

## Section 6: Updating a module inside an application

Another common application management task is to update a module that is contained in an application. In this step, you use the `update()` method of the `AdminApp` object to update the **MyIVTWebApp.war** web module of the MyIVT application with an updated version. This version is packaged in a WAR file named **MyIVTWebAppUpdated.war** and specifies a different display name for the module: **My\_IVT\_Webapp\_Updated**.

You start by using the `help()` method to discover the appropriate invocation syntax and parameter list. You then search the WebSphere Application Server Information Center for an example of how to use the method. Finally, you compose the method, run it, and verify the results by using the administrative console.

- \_\_\_ 1. First, create a script file named **ex04\_updateApp.py** in the **Scripts** folder.
- \_\_\_ 2. Next, use the `help()` method to learn more about the `update()` method. In the `scriptExecutor` Console view, enter:

```
print AdminApp.help("update")
```

```
Please enter a command or script file name to execute, or 'quit' to exit.  
-->print AdminApp.help("update")  
WASX7419I: Method: update      3 parameters  
  
Arguments: application name, content type, options  
  
Description: updates the application specified by "application name" using update type specified by "content type" with options specified by "options".  
  
Valid update content type values are  
    app - entire application  
    file - a single file  
    modulefile - a module These are the option values that you need since you are updating an existing module.  
    partialapp - partial application  
  
All valid options for install command are valid for the update command. Additional update options are  
    -operation - operation to be performed. Valid values are  
        add - adds new contents  
        addupdate - performs add or updates based on existence of contents in the application  
        delete - deletes contents  
        update - update existing contents This option is required if content type is file and modulefile. This option must have "update" as the value if content type is app. This option is ignored if content type is partialapp.  
    -contents - file containing the contents to be added or updated. The specified file has to be local to the scripting client. This option is required except for delete operation.  
    -contenturi - URI of the file to be added, updated, or removed from the application. This option is required if content type is file or modulefile. This option is ignored if content type is app or partialapp.
```

The help for the `update()` method is as follows:

- The method has three parameters: *application name*, *content type*, and *options*.
- *application name* specifies the name of the application to update. Usually, methods that accept this type of argument expect it to be a string that contains the *display name* of the application. In your case, this name is "My\_IVT\_Application"
- *content type* can be one of the listed values. Since you are updating a module inside of an application, the value you must use is `modulefile`.
- *options* provide more information about the *operation*, *contents*, and *contenturi* that are associated with the update.
  - *operation* further specifies the type of update that you are doing. In your case, the value of `update` is what you need since you are updating an existing module.
  - *contents* specifies the name of the file that contains the updated module. In your case, this file is `"/usr/Software/Ears/MyIVTWebAppUpdated.war"`
  - *contenturi* specifies the URI of the file in the EAR file to update. In your case, this file is `"MyIVTWebApp.war"`

- \_\_\_ 3. You now have an idea of what must be specified in the method invocation. But how do you put it together, that is, what is the correct syntax? To find out, look for an example usage in the WebSphere Application Server Information Center.
- \_\_\_ f. In the WebSphere Application Server Information Center browser window (it should still be open from the previous step), find the description of the `update()` method by scrolling down the list of **Commands for the AdminApp object** page. Scroll down to see example syntax of an `update()` method **Using the Jython list** as follows:

Using Jython list:

```
print AdminApp.update('myApp', 'file', ['-operation', 'add', '-contents', 'c:/apps/myApp/web.xml',
'-contenturi', 'META-INF/web.xml'])
```

The example shows that *application name* and *content type* are simple strings, and that *options* are expressed as a list of strings.

- \_\_\_ 4. You are now ready to compose the `update()` method. In the **ex04\_updatedApp.py** editor view, type statements to:
- \_\_\_ a. Assign the wanted values of *application name* and *content type* to variables.
- \_\_\_ b. Assign the wanted values for *operations*, *contents*, and *contenturi* to variables, and combine them into an *options* list.

- \_\_\_ c. Start the `update()` method by passing the variables for *application name*, *content type*, and *options* that are assigned as parameters.
  - \_\_\_ d. Save the configuration change.

Your script should look as follows:



The screenshot shows a Jython code editor window titled "ex04\_updatedApp.py". The code is a script for updating an application named "My\_IVT\_Application" with a module file. It uses the AdminApp.update() method and AdminConfig.save() to perform the update. The code is color-coded, with comments in green and other text in blue.

```
1 #  
2 # TODO: enter JYTHON code and save  
3 #  
4 appName = "My_IVT_Application"  
5 contentType = "modulefile"  
6  
7 # options  
8 operationName = "update"  
9 updatedContentLoc = "/usr/Software/Ears/MyIVTWebAppUpdated.war"  
10 contentURI = "MyIVTWebApp.war"  
11 options = ["-operation", operationName, "-contents", updatedContentLoc, "-contenturi", contentURI]  
12  
13 AdminApp.update(appName, contentType, options)  
14 AdminConfig.save()
```

- \_\_\_ 5. Type **Ctrl+S** to save the script.
  - \_\_\_ 6. Run the `ex04_updateApp.py` script. In the `scriptExecutor` Console view, enter:

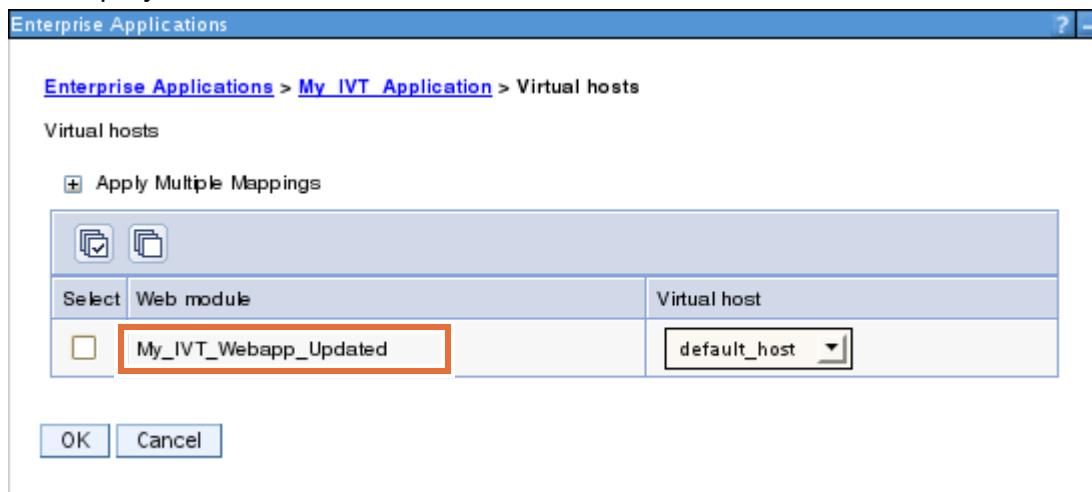
## ex04 updateApp.py

```
ADMA5078I: Update of My_IVT_Application has started.  
ADMA5058I: Application and module versions are validated with versions of deployment targets.  
ADMA5009I: Extracting application archive to /opt/IBM/WebSphere/AppServer/profiles/SamplesProfile/wstemp/appmgmt/mbean/AppManagement_13caac31f3e_1/app_13cab030d94/ext/MyIVTWebApp.war.  
ADMA5064I: FileMergeTask completed successfully for My_IVT_Application.  
ADMA5005I: The application My_IVT_Application is configured in the WebSphere Application Server repository.  
ADMA5005I: The application My_IVT_Application is configured in the WebSphere Application Server repository.  
ADMA5005I: The application My_IVT_Application is configured in the WebSphere Application Server repository.  
ADMA5113I: Activation plan created successfully.  
ADMA5005I: The application My_IVT_Application is configured in the WebSphere Application Server repository.  
ADMA5011I: The cleanup of the temp directory for application My_IVT_Application is complete.  
ADMA5079I: Update of My_IVT_Application has ended. The application or its web modules may require a restart when a save is performed.
```

When execution is finished, a message is shown that states that the update of the application ended.

- \_\_\_ 7. Use the administrative console to verify that the web module was successfully updated. If so, its display name should be **My\_IVT\_Webapp\_Updated**.
  - \_\_\_ a. Log in to the administrative console. In the Admin Console view, type `wasadmin` for **User ID**, `websphere` for **password**, and click **Log in**. The console main page is displayed.
  - \_\_\_ b. In the navigation pane, select **Applications > Application Types > WebSphere enterprise applications**. The list of applications that are installed on the server is displayed in the right pane.
  - \_\_\_ c. In the Enterprise Applications pane, click **My\_IVT\_Application**. The configuration properties of the application are displayed.

- \_\_\_ d. Under Web Module Properties, click **Virtual hosts**. The Virtual hosts pane is displayed.



The Virtual hosts pane shows that the display name of the web module is now **My\_IVT\_Webapp\_Updated**.

- \_\_\_ e. Click **Logout** to exit the administrative console and close the Admin Console view.

You successfully used the AdminApp object to update a module inside an application.

## **Section 7: Cleaning up the environment**

- \_\_\_ 1. Stop the scriptExecutor script. In the scriptExecutor Console window, enter: `quit`.
- \_\_\_ 2. Close any scripts that are still in the editor.
- \_\_\_ 3. Stop the server.
  - \_\_\_ a. Display the Servers view by clicking the **Servers** tab.
  - \_\_\_ b. Make sure the **WebSphere Application Server v8.5 at localhost** server is selected and click **Stop the server**.



Wait until the Servers view displays a status of "Stopped".

- \_\_\_ 4. Exit the IBM Assembly and Deploy Tools by selecting **File > Exit**.

**End of exercise**

## Exercise review and wrap-up

In this exercise, you developed Jython scripts that used the AdminApp object to query, install, and uninstall an application. You also wrote a script to update a module inside an application. You explored the primary methods that are used to perform these tasks and identified the resources that are required to successfully complete them. You ran the resulting Jython scripts in the IBM Assembly and Deploy Tools.



# Exercise 5. Using the AdminControl object

## What this exercise is about

In this exercise, you learn how to use the AdminControl administrative object to query and modify MBean attributes and start MBean operations. Following the general approach for using the AdminControl object, you gain familiarity with the primary methods and resources that are required to effectively use it.

## What you should be able to do

At the end of this exercise, you should be able to:

- Apply the general steps that are required to make an operational or attribute change to an MBean by using the AdminControl object
- Use the primary methods and resources that help in performing each step
- Use the AdminControl object to query and modify MBean attributes, and start MBean operations

## Introduction

The AdminControl administrative object is used for operational control of the WebSphere Application Server environment. It communicates with MBeans that represent running components, and supports methods to query and modify their attributes and start their operations.

When doing problem determination, you often must increase the trace level settings for a server to see more detailed diagnostics messages. In this exercise, you administer the *trace service* component of the *server1* application server by using the AdminControl object. You first query its corresponding MBean, and use it to modify its trace level attribute and then start an operation on it.

The general approach for performing an attribute or operational change by using the AdminControl object is depicted in Figure 1. It

also shows the methods and resources that can be used to help in performing a particular step.

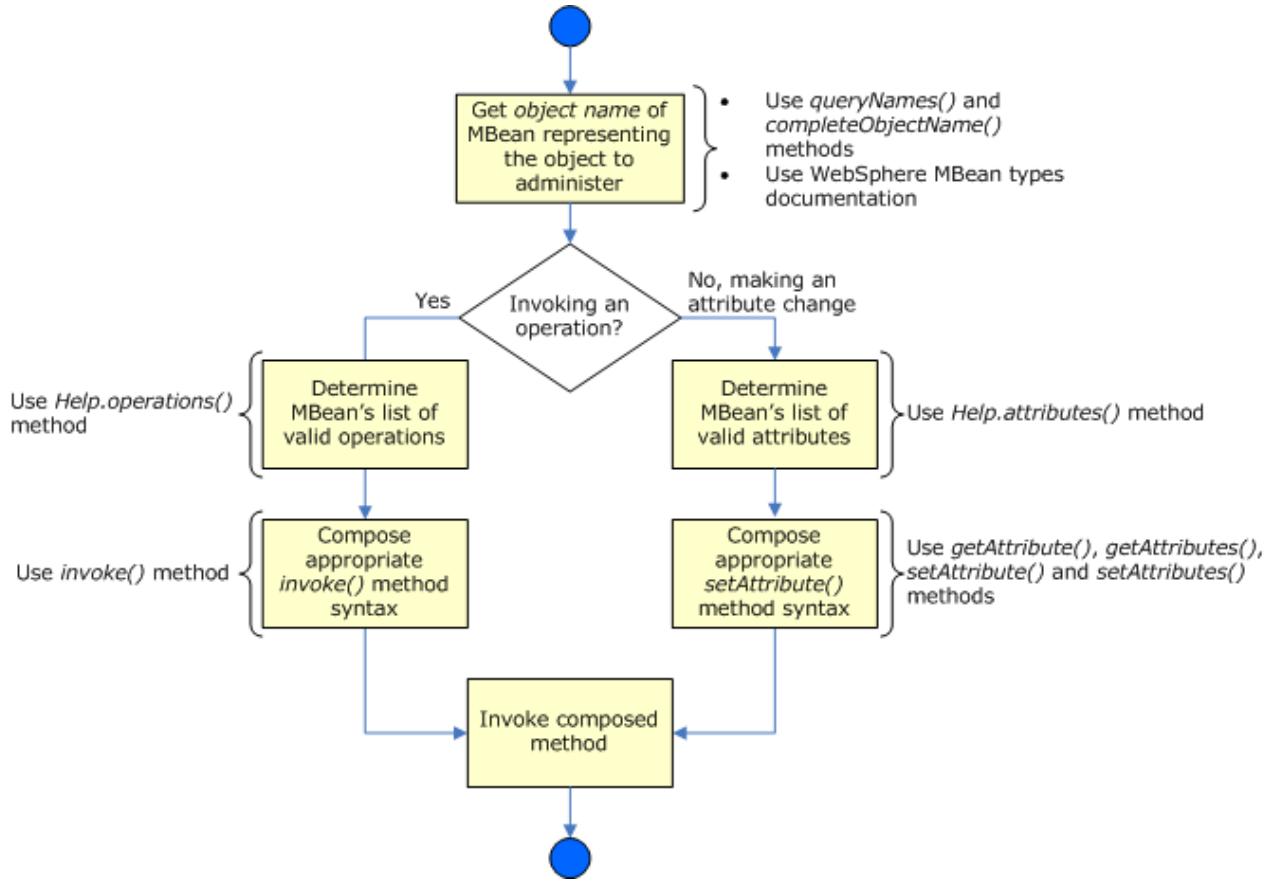


Figure 1 - How to use the AdminControl object

Using the IBM Assembly and Deploy Tools (IADT), you develop Jython scripts that use these primary methods of the AdminControl object.

## Requirements

To complete this exercise, you need the WebSphere Application Server Network Deployment V8.5 product image that is installed and the stand-alone *server1* application server that is created in *SamplesProfile*. In addition, the IBM Assembly and Deploy Tools for WebSphere Administration V8.5 should already be installed.

## Exercise instructions



### Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

`<was_root>`: /opt/IBM/WebSphere/AppServer

`<profile_root>`: /opt/IBM/WebSphere/AppServer/profiles

### Section 1: Preparing the environment

In this step, you open the same IBM Assembly and Deploy Tools (IADT) workspace that you used in the previous exercise, namely, *StandAloneWS*. This workspace already contains a Jython project definition and the definition of the *SamplesProfile server1* stand-alone server. It also already includes the utility script, *scriptExecutor.py*, which is used to facilitate script execution inside the IADT.



### Note

If you have to create or re-create the *StandAloneWS* workspace from scratch, follow the steps that are described in **Section 1: Preparing the environment of Exercise 2**.

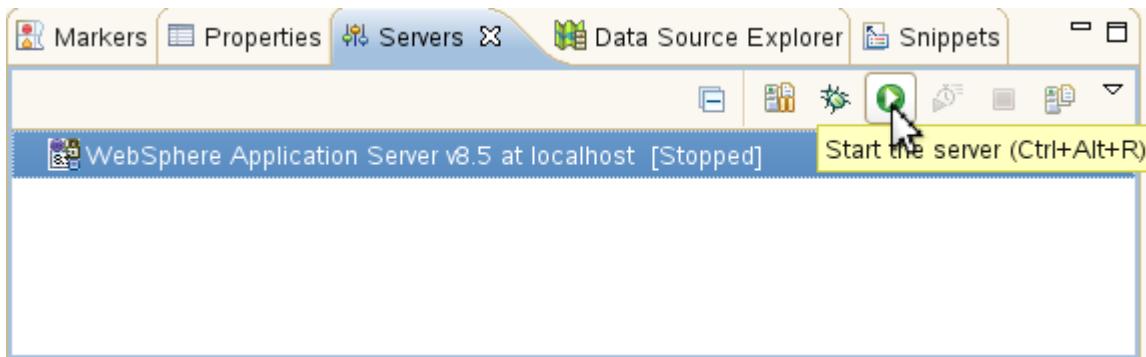
- \_\_\_ 1. Start the IBM Assembly and Deploy Tools and open the **StandAloneWS** workspace.
  - \_\_\_ a. On the desktop, double-click the **IBM Assembly and Deploy Tools** icon.
  - \_\_\_ b. In the Workspace Launcher window, make sure that the Workspace field has a value of /usr/LabWork/StandAloneWS.
  - \_\_\_ c. Click **OK**. The IADT opens and displays the Java EE perspective.
- \_\_\_ 2. Start the **SamplesProfile server1** server.



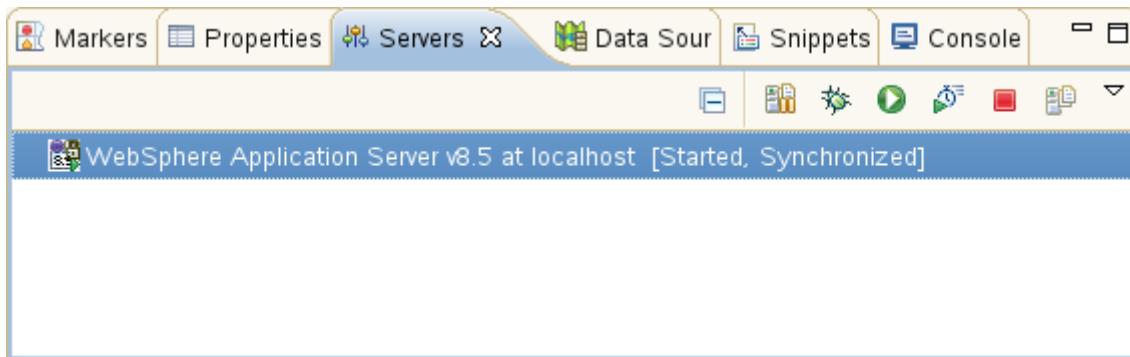
## Information

Using the AdminControl requires a connection to a running server because its methods can be started only on running MBeans.

- \_\_ a. In the Servers view, select **WebSphere Application v8.5 Server at localhost**, and click **Start the server**.

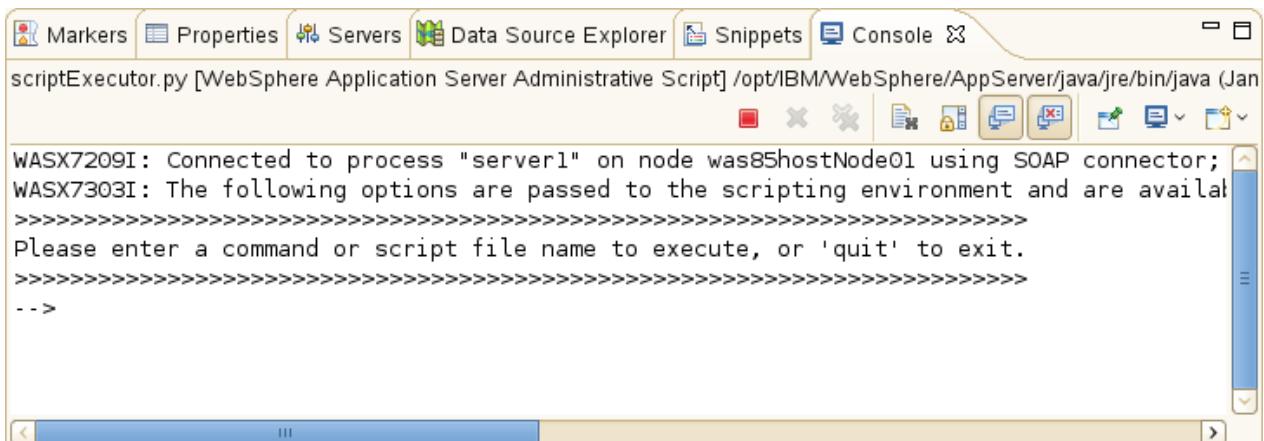


- \_\_ b. Wait until the server is started. As the server starts, focus is switched to the **Console** view where startup messages are displayed. When the server is started, the Servers view displays a status of **Started**.



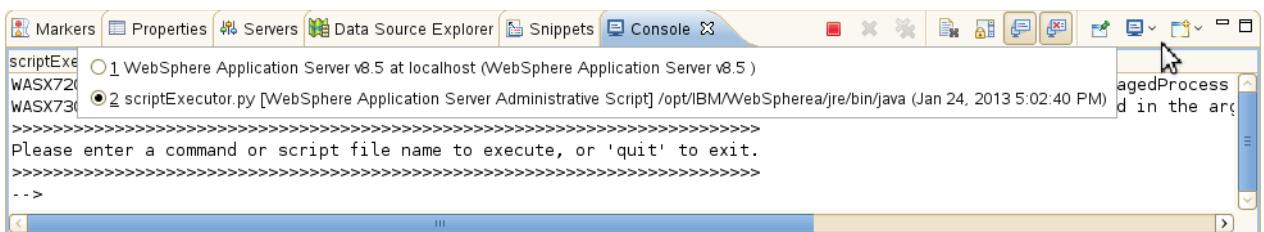
- \_\_ 3. Run the scriptExecutor utility script.
  - \_\_ a. In the Enterprise Explorer view, expand **AdminScriptingProject > Utilities**. Right-click **scriptExecutor.py** and select **Run As > Administrative Script**.
  - \_\_ b. After a few moments, the **Console** view displays a message that confirms the establishment of a wsadmin connection with the *server1* server. An input prompt

is also displayed for running an administrative command or a script file. You are now ready to develop and run your scripts.



## Information

Recall that you have two Console views opened: one for the *scriptExecutor* script and the other of the *server1* standard output. To switch between Console views, click the arrow of the **Display Selected Console** button and select the console output to view.

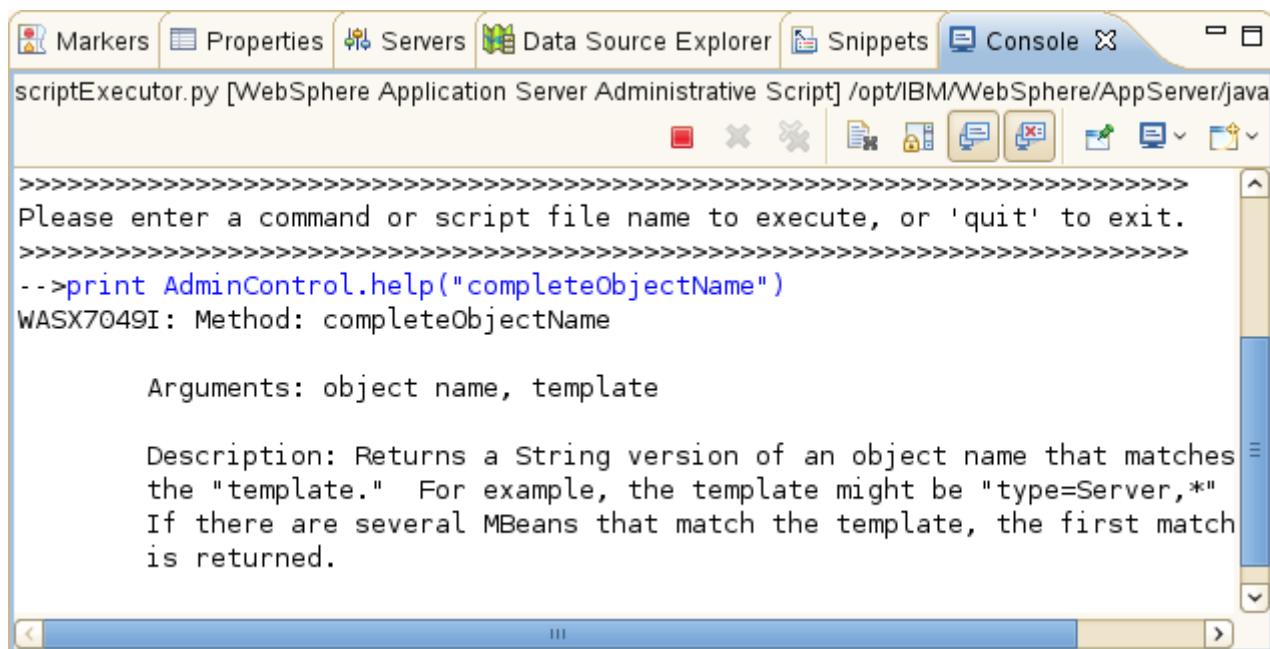


## **Section 2: Getting the object name of the MBean for the trace service**

As illustrated in Figure 1, the first step in using the AdminControl object is to identify the MBean corresponding to the running component to administer and get its *object name*. In this step, you use the *completeObjectName()* method of the AdminControl object to obtain the object name of the MBean that represents the *trace service* in the *server1* application server. In the process, you discover how to use the WebSphere MBean types documentation to identify the list of available MBean types.

- \_\_\_ 1. Start by creating a script file named `ex05_getObjectName.py` in the Scripts folder.
    - \_\_\_ a. In the Enterprise Explorer view, right-click the **Scripts** folder and select **New > Other**. The Select a wizard dialog opens.
    - \_\_\_ b. In the **Select a wizard** dialog, expand **Jython** and select **Jython Script File**. Click **Next**.
    - \_\_\_ c. In the **Create a Jython script file** dialog, type `ex05_getObjectName.py` in the File name field.
    - \_\_\_ d. Click **Finish**. The new script file is displayed in the Enterprise Explorer view in the Scripts folder, and is opened in the Jython editor view.
  - \_\_\_ 2. Next, get help on how to use the `completeObjectName()` method. In the scriptExecutor Console view, enter:

```
print AdminControl.help("completeObjectName")
```





### Note

The help output for the `completeObjectName()` method actually has a typographical error. The comma character in the `Arguments` list should not be there, as it suggests that the method takes two arguments. Without the comma, the output correctly specifies that only one argument is required, an *object name template*.

The help output indicates that the method requires an *object name template* as a parameter. An object name template is a string that contains a portion of an object name and is used for pattern matching when searching for actual object names. It has the following format:

`"domainName:keyProperty1=value1,keyProperty2=value2,..."`

Where `domainName` is always `WebSphere` and key properties include:

- `type`: Type of MBean associated with object
- `name`: Display name of object
- `node`: Name of node where object runs
- `process`: Name of server process in which object runs

Specifying more key property values allows you to narrow the scope of the search. The key property `type` is almost always specified in an object name template as a starting point for a search.

- \_\_\_ 3. What is the MBean type that corresponds to a trace service object? You can use the MBean type documentation in the WebSphere Application Server V8.5 Information Center to find out.
  - \_\_\_ a. If you have Internet access, open a browser and enter the following web address:  
`http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.javadoc.doc%2Fweb%2FmbeanDocs%2FTraceservice.html`
  - \_\_\_ b. If you do not have Internet access, a screen capture of the Information Center page is shown here.

- \_\_\_ c. The documentation page is shown in the following screen capture..

## TraceService MBean

**Partial ObjectName:**  
WebSphere:\*, type=TraceService

MBean TraceService

Management interface for the live TraceService function running in a server.

<b>Attribute Summary</b>	
int	<a href="#">ringBufferSize</a> the size of the trace output Ring Buffer.
java.lang.String	<a href="#">traceSpecification</a> A Trace specification which allows enabling/disabling tracing for components.
java.lang.String	<a href="#">effectiveTraceSpecification</a> A Trace specification like traceSpecification only with external factors such as PCI-DSS applied.
java.lang.String	<a href="#">traceFileName</a> the name of the file to which trace output will be written.
java.lang.String	<a href="#">traceRuntimeConfig</a> The current runtime configuration of the trace logger.
boolean	<a href="#">rawTraceFilterEnabled</a> Raw Trace Filter Enablement for legacy.

You see that this MBean type represents a "Management interface for the live TraceService function running in a server". It is the MBean type for a trace service object.

- \_\_\_ 4. You are now ready to compose the method to retrieve the object name of the **TraceService** MBean on **server1** application server. In the editor view, type the following command on one line:

```
print
AdminControl.completeObjectName ("WebSphere:type=TraceService,process=
server1,*")
```

**Hint**

Unless you know the value of all of the key properties for an object, make sure to always end the template with the wildcard (\*) value. Omitting it causes the method to look for a match with exactly the specified key properties and no others, which often is not the intent.

```

SPV *ex05_getObjectName.py < ...
1 #
2 # TODO: enter JYTHON code and save
3 #
4 print AdminControl.completeObjectName("WebSphere:type=TraceService,process=server1,*")

```

- \_\_\_ 5. Type **Ctrl+S** to save the script.
- \_\_\_ 6. Run the **ex05\_getObjectName.py** script. In the scriptExecutor Console view, enter:

**ex05\_getObjectName.py**

Markers Properties Servers Data Source Explorer Snippets Console < ...

scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/java/jre/bin/java (Feb 5, 2013)

-->ex05\_getObjectName.py

WebSphere:name=TraceService,process=server1,platform=proxy,node=was85hostNode01,version=8.5.0.0,type=TraceService,mbeanIdentifier=cells/was85hostNode01Cell/nodes/was85hostNode01 servers/server1/server.xml#TraceService\_1183122130078,cell=was85hostNode01Cell,spec=1.0

The object name of the *TraceService* MBean for *server1* is returned and displayed.

**Information**

To wrap the lines of the Console view at 80 characters per line:

- Right-click anywhere inside the view and select **Preferences**.
- In the Preferences window, select the check box for **Fixed width console**. The maximum character width is already preset to 80.
- Click **OK**.



## Information

If more than one MBean matches the supplied object name template, the default behavior of the *completeObjectName()* method is to return the first match. To retrieve all matches in a result string, use the *queryNames()* method. For example, to display the names of all DataSource MBean objects, enter the following in the scriptExecutor Console view:

```
print AdminControl.queryNames ("WebSphere:type=DataSource,*")
```

You successfully retrieved the object name of the *TraceService* MBean on server1.

## Section 3: Querying and modifying the attributes of a TraceService MBean

You are now ready to explore the AdminControl methods that allow you to query and modify an MBean attribute. In this step, you discover the attributes of the server1 TraceService MBean and change the one that controls its trace level. Specifically, you use the Help administrative object to obtain help on identifying MBean attributes, and the AdminControl *getAttribute()* and *setAttribute()* methods to query and modify the trace level attribute.

- \_\_\_ 1. Create another script file named `ex05_modifyTraceMBeanAttribute.py` in the Scripts folder.
- \_\_\_ 2. What is the name of the TraceService MBean attribute that specifies the trace level?
  - \_\_\_ a. You can use the *attributes(anObjectName)* method of the Help object to find out. This method accepts an *object name* as parameter and displays the attribute list of the MBean identified by it. In the editor view for `ex05_modifyTraceMBeanAttribute.py`, enter:

```
traceServiceObjectName =
AdminControl.completeObjectName("WebSphere:type=TraceService,process=server1,*")
print Help.attributes(traceServiceObjectName)
```

```
*ex05_modifyTraceMBeanAttribute.py
1 #
2 # TODO: enter JYTHON code and save
3 #
4 traceServiceObjectName = AdminControl.completeObjectName("WebSphere:type=TraceService,process=server1,*")
5 print Help.attributes(traceServiceObjectName)
```

- \_\_\_ b. Type **Ctrl+S** to save the script.
- \_\_\_ c. Run the `ex05_modifyTraceMBeanAttribute.py` script. In the scriptExecutor Console view, enter:

ex05\_modifyTraceMBeanAttribute.py

```

Markers Properties Servers Data Source Explorer Snippets Console
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/java
Please enter a command or script file name to execute, or 'quit' to exit.
Or press the 'Enter' key to execute the ex05_getObjectName.py script.
-->ex05_modifyTraceMBeanAttribute.py
Attribute          Type          Access
ringBufferSize      int           RW
traceSpecification java.lang.String  RW
effectiveTraceSpecification java.lang.String  RO
traceFileName       java.lang.String  RO
traceRuntimeConfig  java.lang.String  RO
rawTraceFilterEnabled boolean        RW

```

The method returns the name, type and access policy of all of the attributes of the TraceService MBean. Notice the attribute named `traceSpecification`. It sounds like the one used to specify the trace level.

- \_\_\_ d. To verify this attribute, look at the WebSphere MBean type documentation in the Information Center for the **TraceService MBean** type. In the Attributes Summary section, read the description of the `traceSpecification` attribute.

## Attribute Summary

int	<a href="#">ringBufferSize</a> the size of the trace output Ring Buffer.
java.lang.String	<a href="#">traceSpecification</a> A Trace specification which allows enabling/disabling tracing for components.

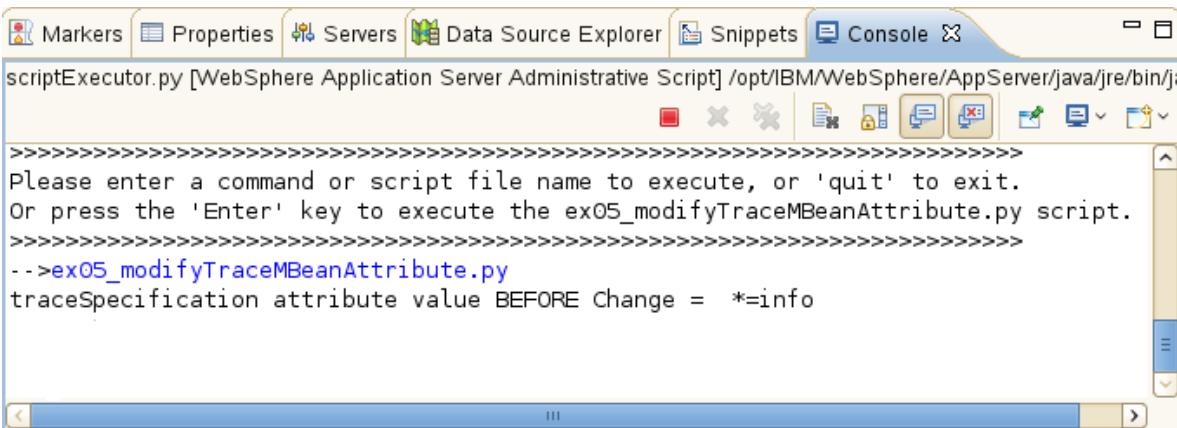
The description validates the fact that `traceSpecification` is the attribute to use for specifying the trace service component trace level.

- \_\_\_ 3. Now that you identified the MBean attribute of interest, what is its current value? Use the `getAttribute()` method to find out.
  - \_\_\_ a. In the editor view:
    - Comment out the line that prints the attribute list of the TraceService MBean.
    - Add statements to print the "before" value of the `traceSpecification` attribute.

Your script should look as follows:

```
*ex05_modifyTraceMBeanAttribute.py <input>
1 #
2 # TODO: enter JYTHON code and save
3 #
4 traceServiceObjectName = AdminControl.completeObjectName("WebSphere:type=TraceService,process=server1,*")
5 #print Help.attributes(traceServiceObjectName)
6 print "traceSpecification attribute value BEFORE Change =", 
7 print AdminControl.getAttribute(traceServiceObjectName, "traceSpecification")
```

- b. Type **Ctrl+S** to save the script.
  - c. Run the script. In the scriptExecutor Console view, press Enter.



The output shows that the initial value of traceSpecification is "`*=info`".

- \_\_\_ 4. You are now ready to compose a *setAttribute()* method to increase the trace level to "*\*=detail*". In the Editor view:

\_\_\_ a. Add the following line:

```
AdminControl.setAttribute(traceServiceObjectName,  
"traceSpecification", "*=detail")
```

\_\_\_ b. Add statements to print the "after" value of the *traceSpecification* attribute.

Your script should look as follows:

```
*ex05_modifyTraceMBeanAttribute.py x
1 #
2 # TODO: enter JYTHON code and save
3 #
4 traceServiceObjectName = AdminControl.completeObjectName("WebSphere:type=TraceService,process=server1,*")
5 #print Help.attributes(traceServiceObjectName)
6
7 print "traceSpecification attribute value BEFORE Change = ",
8 print AdminControl.getAttribute(traceServiceObjectName, "traceSpecification")
9
10 # Modify traceSpecification attribute
11 AdminControl.setAttribute(traceServiceObjectName, "traceSpecification", "*=detail")
12
13 print "traceSpecification attribute value AFTER Change = ",
14 print AdminControl.getAttribute(traceServiceObjectName, "traceSpecification")
```

- \_\_\_ 5. Type **Ctrl+S** to save the script.
  - \_\_\_ 6. Run the script. In the scriptExecutor Console view, press Enter.

The output shows that the value of `traceSpecification` was successfully changed to "`*=detail`".



## Note

The console view might automatically change to the server1 view. If not, select the server1 console view and verify that you see a message that shows the new trace state.

[3/6/13 10:43:31:635 EST] 00000051 ManagerAdmin I TRAS0018I: The trace state has changed. The new trace state is \*=detail.

- \_\_\_ 7. As an extra check, use the administrative console to verify that the trace level for *server1* was properly modified.

\_\_\_ a. Display the Servers view by clicking the **Servers** tab.

\_\_\_ a. Right-click **WebSphere Application Server v8.5 at localhost**, and select **Run administrative console**.

- \_\_\_ b. In the Security Alert window, click **OK** to accept the security certificate and proceed.
- \_\_\_ c. The administrative console opens in the Admin Console view and displays the Login page. Type `wasadmin` for **User ID**, `web1sphere` for **password**, and click **Log in**.
- \_\_\_ d. The main page of the Administrative console is displayed. In the navigation pane, select **Troubleshooting > Logs and Trace**. The Logging and Tracing page is displayed in the right pane.
- \_\_\_ e. In the right pane, select **server1**. The General Properties page for server1 is displayed.
- \_\_\_ f. In the General Properties page, select **Change Log Detail Levels**. The **Configuration** log detail level page is displayed.

The screenshot shows the 'Logging and tracing' interface in the Admin Console. The 'Runtime' tab is active. In the 'General Properties' section, there's a 'Change log detail levels' area with a text input field containing the value `*=info`. This input field is highlighted with a red border.

Notice that the log level indicated on the **Configuration** tab was *not changed*. This result is because updates to attributes of an MBean are transient and do not affect the persistent configuration of the object that it represents.

- \_\_\_ g. Select the **Runtime** tab. The **Runtime** log detail level page is displayed.

The screenshot shows the 'Logging and tracing' interface with the 'server1' server selected. The 'Runtime' tab is highlighted with a red box. Under 'General Properties', there is a checkbox for 'Save runtime changes to configuration as well'. The 'Change log detail levels' section contains a checkbox for 'Disable logging and tracing of potentially sensitive data (WARNING: This might cause the log detail level setting to be modified when it is applied on the server.)'. Below this, instructions say to 'Select components and specify a log detail level. Log detail levels specified here will apply to the entire server. Expand Components and Groups and click Components to specify a log detail level for individual components, or click Groups to specify a log detail level for a predefined group of components. Click a component or group name to select a log detail level. Log detail levels are cumulative.' A text input field contains the value '\*=detail', which is also highlighted with a red box. A link labeled '+ Components and Groups' is visible below the input field.

Here you see that the log level is changed to "`*=detail`".

- \_\_\_ h. Click **Logout** to exit the administrative console and close the Admin Console view.

You successfully used the AdminControl object to change the trace level setting for the `server1` server.

## Section 4: Starting an operation on a TraceService MBean

You can also use the AdminControl object to start MBean operations. In this step, you discover the available operations for the *TraceService* MBean and start one to append a trace string to the active trace state. The additional trace string enables maximum tracing on all "com.ibm.\*" components.

- \_\_\_ 1. Create another script file named `ex05_invokeTraceMBeanOperation.py` in the Scripts folder.
- \_\_\_ 2. What operations can be started on the *TraceService* MBean?
  - \_\_\_ a. You can use the variations of the Help object `operations()` method to find out. In the editor view for `ex05_invokeTraceMBeanOperation.py`, enter:

```
traceServiceObjectName =
AdminControl.completeObjectName ("WebSphere:type=TraceService,process=server1,*")
print Help.operations (traceServiceObjectName)
```

The screenshot shows a Jython script editor window titled "SPY \*ex05\_invokeTraceMBeanOperation.py". The code in the editor is:

```
1 #
2 # TODO: enter JYTHON code and save
3 #
4 traceServiceObjectName = AdminControl.completeObjectName("WebSphere:type=TraceService,process=server1,*")
5 print Help.operations(traceServiceObjectName)
```

- \_\_\_ b. Type **Ctrl+S** to save the script.
- \_\_\_ c. Run the `ex05_invokeTraceMBeanOperation.py` script. In the scriptExecutor Console view, enter:

## ex05 invokeTraceMBeanOperation.py

The method returns the type and name of all of the operations available for the `TraceService` MBean. Notice one called `appendTraceString()`. Find out more about it in the next.

- \_\_\_ d. To further determine what each operation does and its invocation syntax, use the *operations(anObjectName, anOperationName)* variation of the method. In the editor view:

- Comment out the line for the first `Help.operations()` method invocation.
  - Add the following line:

```
print Help.operations(traceServiceObjectName, "appendTraceString")
```



The screenshot shows a Jython script editor window with the following code:

```
1 #  
2 # TODO: enter JYTHON code and save  
3 #  
4 traceServiceObjectName = AdminControl.completeObjectName("WebSphere:type=TraceService,process=server1,*")  
5 #print Help.operations(traceServiceObjectName)  
6 print Help.operations(traceServiceObjectName, "appendTraceString")
```

- \_\_ e. Type **Ctrl+S** to save the script.

- \_\_\_ f. Run the ex05\_invokeTraceMBeanOperation.py script. In the scriptExecutor Console view, press Enter.

```

Markers Properties Servers Data Source Explorer Snippets Console < >
scriptExecutor.py [WebSphere Application Server Administrative Script] /opt/IBM/WebSphere/AppServer/java/jre/bin/java (Fe
Please enter a command or script file name to execute, or 'quit' to exit.
Or press the 'Enter' key to execute the ex05_invokeTraceMBeanOperation.py script.
-->
void appendTraceString(java.lang.String)

Description: Add the specified trace string to the trace state already active in the
process runtime, without first clearing the current state.

Parameters:

Type java.lang.String
Name traceString
Description a String that complies to the TraceString grammar and passes the checkTr
aceString() method.

```

The method returns a description of the `appendTraceString` operation for the **TraceService MBean**, including its parameter list. Specifically, it indicates that the operation takes one parameter, a `traceString`, and adds it to the current active trace state without first clearing it.

- \_\_\_ g. Look at the WebSphere MBean type documentation in the Information Center for the **TraceService MBean** type. Scroll down to the **Operations Summary** section to see the list of all supported operations and their description.

## Operation Summary

<code>void <a href="#">appendTraceString</a>(java.lang.String traceString)</code>	Add the specified trace string to the trace state already active in the process runtime, without first clearing the current state.
<code>void <a href="#">dumpRingBuffer</a>(java.lang.String fileToWriteTo)</code>	Write the contents of the Ras services Ring Buffer to the specified file.

By clicking an operation name link, you can get further detail about its usage.

- \_\_\_ 3. Now, compose an `invoke()` method to append the following trace string to the current trace level setting: "com.ibm.\*=all=enabled".
- \_\_\_ a. Begin by getting help on the usage of the `invoke()` method. In the `scriptExecutor` Console view, type:

```
print AdminControl.help("invoke")
```

The help output indicates several invocation forms of the `invoke()` method. Since you are starting an operation that requires a parameter, the second form is the one to use.

- \_\_\_ b. In the editor view:

  - Comment out the line for the second `Help.operations()` method invocation.
  - Add statements to print the "before" value of the MBean `traceSpecification` attribute.
  - Add the following line:

```
AdminControl.invoke(traceServiceObjectName, "appendTraceString",  
"com.ibm.*=all")
```

- Add statements to print the "after" value of the MBean traceSpecification attribute.

Your script should look as follows:

```
SPV *ex05_invokeTraceMBeanOperation.py <input>
1 #
2 # TODO: enter JYTHON code and save
3 #
4 traceServiceObjectName = AdminControl.completeObjectName("WebSphere:type=TraceService,process=server1,*")
5 #print Help.operations(traceServiceObjectName)
6 #print Help.operations(traceServiceObjectName, "appendTraceString")
7
8 print "traceSpecification attribute value BEFORE change = ",
9 print AdminControl.getAttribute(traceServiceObjectName, "traceSpecification")
10
11 # Modify traceSpecification
12 AdminControl.invoke(traceServiceObjectName, "appendTraceString", "com.ibm.*=all")
13
14 print "traceSpecification attribute value AFTER change = ",
15 print AdminControl.getAttribute(traceServiceObjectName, "traceSpecification")
```

- \_\_\_ c. Type **Ctrl+S** to save the script.
  - \_\_\_ 4. Run the **ex05\_invokeTraceMBeanOperation.py** script. In the scriptExecutor Console view, press Enter.

- \_\_ a. If the console view does not automatically switch to server1, select the server1 console view and verify that you see a message such as:

ManagerAdmin I TRAS0018I: The trace state has changed. The new trace state is \*=detail:com.ibm.\*=all.

- \_\_\_ b. Log in to the administrative console as you did earlier. Select **Troubleshooting > Logging and tracing > server1 > Change log detail levels**. Verify that the trace specification is updated to `*=detail:com.ibm.*=all` on the Runtime tab.
  - \_\_\_ c. Log out of the administrative console.

Success! The execution output shows that

successfully used the AdminControl object to start an MBean operation.

## **Section 5: Cleaning up the environment**

- \_\_\_ 1. Stop the scriptExecutor script. In the scriptExecutor Console window, enter: `quit`.
- \_\_\_ 2. Close any scripts that are still open in the Jython editor.
- \_\_\_ 3. Stop the server.
  - \_\_\_ a. Display the Servers view by clicking the **Servers** tab.
  - \_\_\_ b. Make sure the **WebSphere Application Server v8.5 at localhost** server is selected and click **Stop the server**.



Wait until the Servers view displays a status of "Stopped".

- \_\_\_ 4. Exit the IBM Assembly and Deploy Tools by selecting **File > Exit**.

**End of exercise**

## Exercise review and wrap-up

In this exercise, you used the AdminControl object to query and modify MBean attributes and start MBean operations. You followed the steps for performing an MBean attribute or operational change with the AdminControl object and identified the primary methods and resources that are required to successfully complete each step. You developed and ran the resulting Jython scripts in the IBM Assembly and Deploy Tools.



# Exercise 6. Using the AdminTask object

## What this exercise is about

In this exercise, you learn how to perform administrative tasks by using the AdminTask object. Following the general approach for using the AdminTask object, you gain familiarity with the primary methods and resources that are required to effectively use it.

## What you should be able to do

At the end of this exercise, you should be able to:

- Apply the general steps that are required to start a command in batch or interactive mode by using the AdminTask object
- Use the primary methods and resources that help in performing each step
- Use the AdminTask object to perform administrative tasks in interactive or batch mode

## Introduction

The AdminTask administrative object provides a set of task-oriented commands for performing administrative functions. It was added to the list of wsadmin administrative objects in WebSphere Application Server V6.0 with the purpose of providing a more user-friendly way to perform scripted administration. To use the AdminConfig object, for example, requires understanding the WebSphere configuration model, while using AdminControl requires knowledge of the available MBean types. In contrast, using AdminTask is much easier because it provides commands that directly represent administrative actions that a user performs.

In the previous exercise, you used the AdminControl object to modify the trace level setting for the stand-alone server, *server1*. This task involved a multi-step process that required you to identify the MBean that represented the *TraceService* of the server, get its object name, and understand its attributes and operations. In this exercise, you accomplish this same task in a much easier way by using the AdminTask object.

The general steps to follow when using the AdminTask object is depicted in Figure 1. It also shows the resources that can be used to help in performing a particular step.

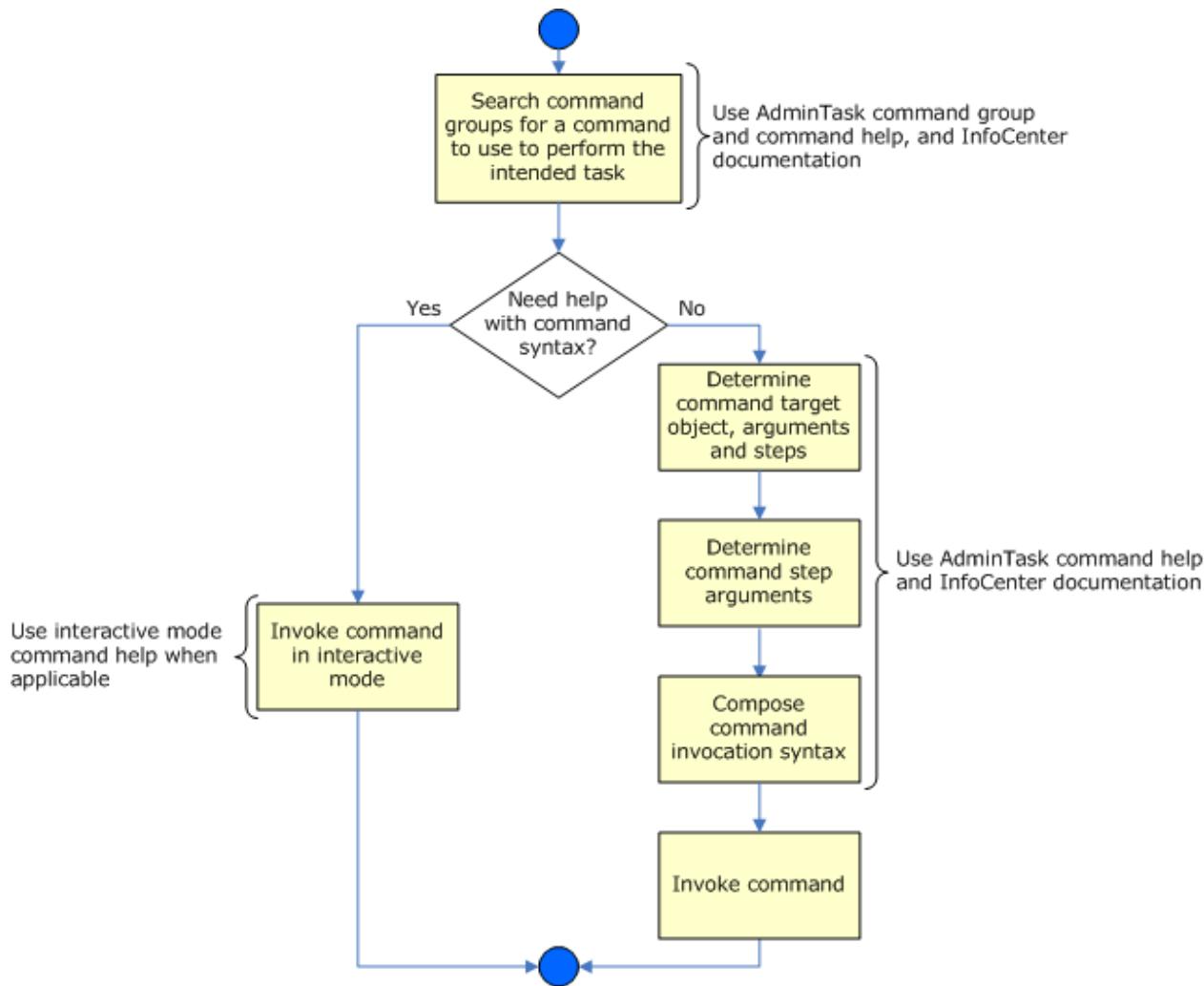


Figure 1 - How to use the AdminTask object

Using the IBM Assembly and Deploy Tools (IADT), you run the appropriate AdminTask command in interactive mode. You also develop a Jython script to run it in batch mode.

## Requirements

To complete this exercise, you need the WebSphere Application Server Network Deployment V8.5 product image that is installed and the stand-alone *server1* application server that is created in *SamplesProfile*. In addition, the IBM Assembly and Deploy Tools v8.5 must already be installed.

## Exercise instructions



### Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

`<was_root>`: /opt/IBM/WebSphere/AppServer

`<profile_root>`: /opt/IBM/WebSphere/AppServer/profiles

### Section 1: Preparing the environment

In this step, you open the *StandAloneWS* workspace that you used in the previous exercise in the IBM Assembly and Deploy Tools (IADT). This workspace already contains a Jython project definition and the definition of the *SamplesProfile server1* stand-alone server. It also already includes the utility script, *scriptExecutor.py*, which is used to facilitate script execution inside the IADT.



### Note

If you must create or re-create the *StandAloneWS* workspace from scratch, follow the steps that are described in **Section 1: Preparing the environment** of **Exercise 2**.

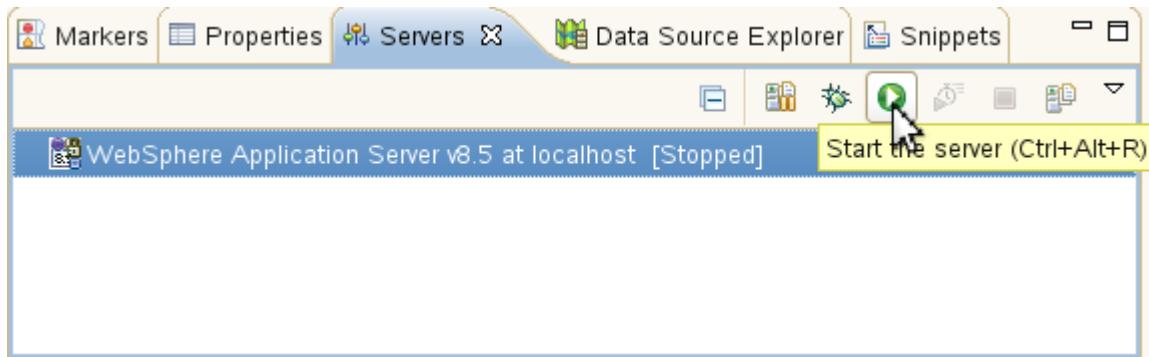
- \_\_\_ 1. Start the IBM Assembly and Deploy Tools and open the **StandAloneWS** workspace.
  - \_\_\_ a. On the desktop, double-click the **IBM Assembly and Deploy Tools** icon.
  - \_\_\_ b. In the Workspace Launcher window, make sure that the Workspace field has a value of /usr/LabWork/StandAloneWS.
  - \_\_\_ c. Click **OK**. The IADT opens and displays the Java EE perspective.
- \_\_\_ 2. Start the **SamplesProfile server1** server.



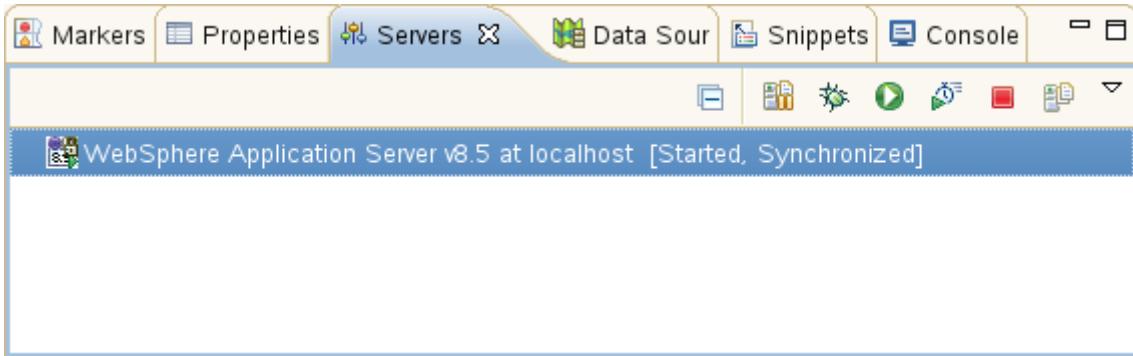
## Information

Using the AdminTask object does not require a connection to a running server. Therefore, the script that is developed in this lab can run without being connected to *SamplesProfile server1*. However, because you open the administrative console later to verify your configuration changes, you are asked to start the server now for convenience.

- \_\_ a. In the Servers view, select **WebSphere Application v8.5 Server at localhost**, and click the **Start the server** button.

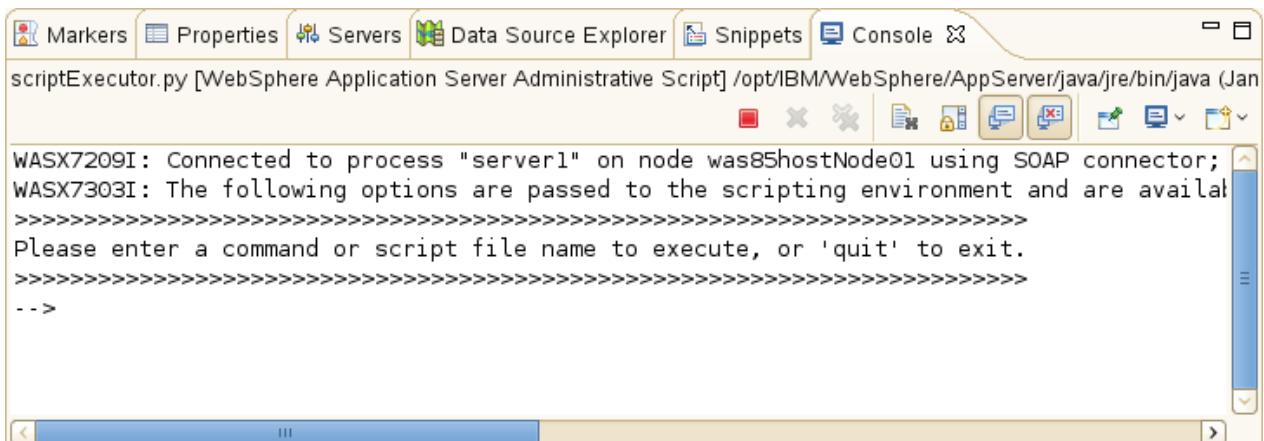


- \_\_ b. Wait until the server is started. As the server starts, focus is switched to the **Console** view where startup messages are displayed. When the server is started, the Servers view displays a status of **Started**.



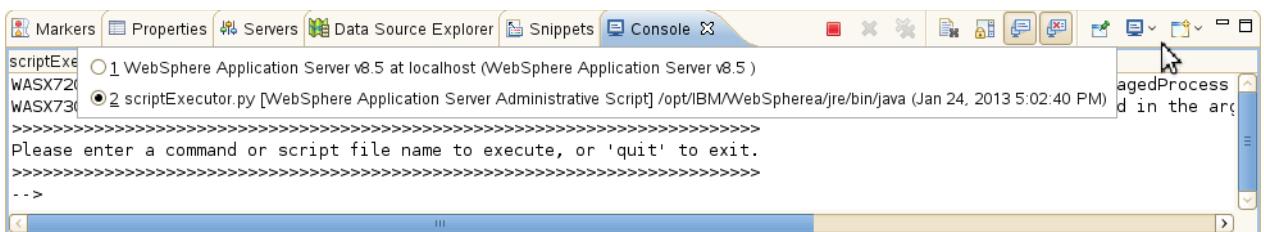
- \_\_ 3. Run the **scriptExecutor** utility script.
  - \_\_ a. In the Enterprise Explorer view, expand **AdminScriptingProject > Utilities**. Right-click **scriptExecutor.py** and select **Run As > Administrative Script**.
  - \_\_ b. After a few moments, the Console view displays a message that confirms the establishment of a wsadmin connection with the *server1* server. An input prompt

is also displayed for running an administrative command or a script file. You are now ready to develop and run your scripts.



## Information

Recall that you have two Console views opened: one for the *scriptExecutor* script and the other of the *server1* standard output. To switch between Console views, click the drop-down arrow of the **Display Selected Console** and select the console output to view.



## ***Section 2: Identifying the command to use***

As illustrated in Figure 1, the first step in using the AdminTask object is to identify the command to use to perform the intended task. In this step, you use several help methods and the WebSphere Information Center documentation to find the name of the AdminTask command that allows you to change the trace specification for a server.

- \_\_\_ 1. Begin by creating a script file named `ex06_setTraceLevel.py` in the Scripts folder.
    - \_\_\_ a. In the Enterprise Explorer view, right-click the **Scripts** folder and select **New > Other**. The Select a wizard dialog opens.
    - \_\_\_ b. In the **Select a wizard** dialog, expand **Jython** and select **Jython Script File**. Click **Next**.
    - \_\_\_ c. In the **Create a Jython script file** dialog, type `ex06_setTraceLevel.py` in the File name field.
    - \_\_\_ d. Click **Finish**. The new script file is shown in the Enterprise Explorer view in the Scripts folder, and is opened in the Jython editor view.
  - \_\_\_ 2. Start your search by listing all the commands available for the AdminTask object by using the `help("-commands")` method. In the scriptExecutor Console view, enter:

```
print AdminTask.help("-commands")
```

The help output returns a list of over 550 available commands along with their description! Although this information might be useful if you already know some part of the command name, it is impractical to sequentially scroll down this list to find the appropriate command when you are starting from scratch.

- \_\_ 3. A better search approach is to look at the available AdminTask command groups that combine commands by functionality. If you can deduce or guess which group

contains the command that you are looking for, you can narrow your search to only the commands in that group.

- \_\_ a. Use the `help("-commandGroups")` method list all the available command groups. In the `scriptExecutor` Console view, enter:

```
print AdminTask.help("-commandGroups")
```

This time the returned list of command groups is shorter. Scroll down the list to try to find a group that can contain a command to change the trace specification of the server that is based on its description. Can you find one or more candidates?

- \_\_ b. As you find a candidate, use the `help(aCommandGroupName)` method to list the commands that it contains. For example, to list the commands in the *ResourceManagement* command group (a potential candidate in your case?), in the scriptExecutor Console view, enter:

```
print AdminTask.help("ResourceManagement")  
Please enter a command or script file name to execute, or 'quit' to exit.  
-->print AdminTask.help("ResourceManagement")  
WASX8007I: Detailed help for command group: ResourceManagement  
  
Description: Group of command that manages resources.  
  
Commands:  
setResourceProperty - This command sets the value of a specified property defined on a resource provider such as JDBCProvider or a connection factory such as DataSource or JMSConnectionFactory. If the property with specified key is defined already, then this command overrides the value. If none property with specified key is defined yet, then this command will add the property with specified key and value.  
  
showResourceProperties - This command lists all the property values defined on a resource provider such as JDBCProvider or a connection factory such as DataSource or JMSConnectionFactory.
```

By reading their description, you can determine that the listed commands all deal with manipulating the properties of a resource provider. Therefore, the command you are looking for is not in this group.

You can repeat this process with other command group candidates. However, as with the previous approach, unless you are already familiar with many of the listed command groups, the search can be tedious and difficult.

- 4. The third resource that you can use to find the AdminTask command appropriate for your task is the WebSphere Application Server V8.5 Information Center. Open a browser to it and use its search function to try and find a match:

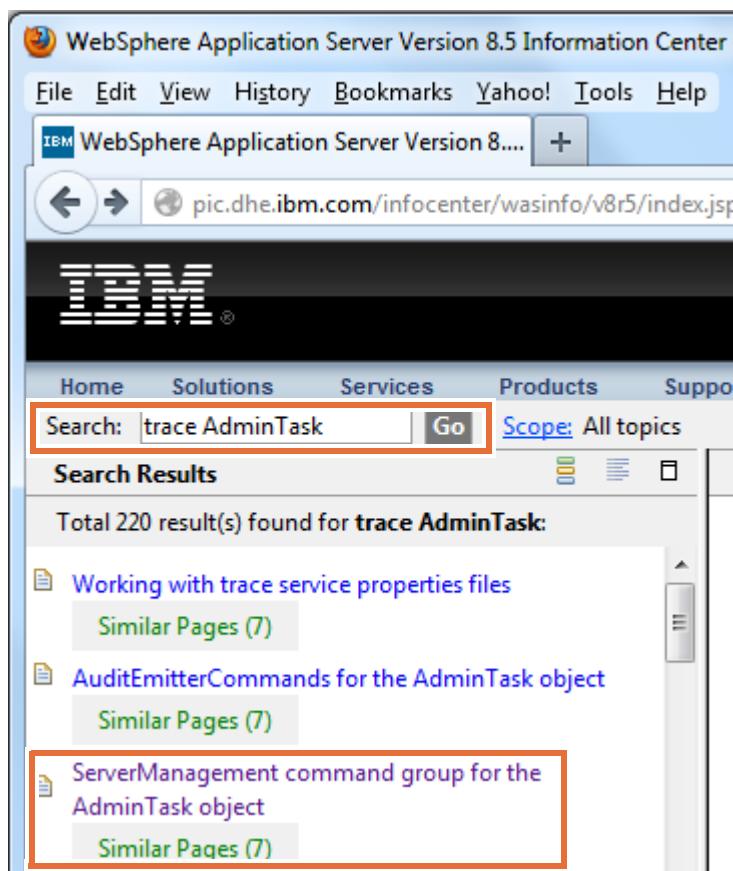


### Important

If you do not have Internet access, there is no problem. All of the search results are included in this exercise as screen captures.

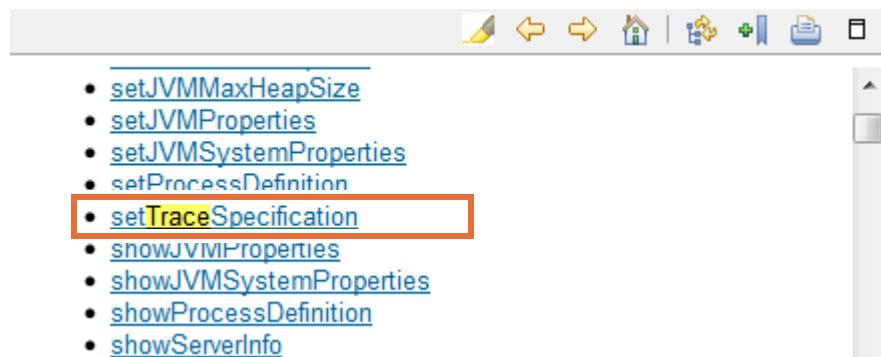
- a. If you have Internet access, open a web browser and enter the following web address:  
<http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp>
- b. What should you search for? Using some intuition can help greatly. In your case, you are looking for an *AdminTask* command that allows you to set the *trace* level

of the server. In the Search box, type **trace AdminTask** and click **GO**. The search returns the list of hits in a Search Results pane.



Among the returned hits, the group of entries, *ServerManagement command group for the AdminTask object*, seems to be the best match. It makes sense for setting a trace level to be categorized as a server management task.

- c. In the Search Results pane, click **Similar Pages (7)** under the match named **ServerManagement command group for the AdminTask object** and select the link that is associated with the *Network Deployment (Distributed operating systems), Version 8.5*. The *ServerManagement command group for the AdminTask object* page is displayed in the right pane, and the view is positioned on the first match for the **trace** keyword: *setTraceSpecification*.



- \_\_\_ d. Can this command be the one you are looking for? To find out, click the **setTraceSpecification** link. The view is positioned to show the description of the command.

**setTraceSpecification**

Use the **setTraceSpecification** command to set the **trace** specification for the server. If the server is running new **trace** specification the change takes effect immediately. This command also saves the **trace** specification in configuration.

**Target object**

None

Reading the description of the `setTraceSpecification` command confirms that it is the one that you are looking for.

Leave the Information Center browser window open for later use.

Success! You identified the AdminTask command to use for setting the trace level of a server: the `setTraceSpecification` command in the `ServerManagement` command group.

### ***Section 3: Starting the command in interactive mode***

What is the easiest way to learn how to use an AdminTask command? The answer is to start it in interactive mode. This execution mode provides a text-based wizard that allows you to enter the required parameters interactively and generates the full invocation syntax for eventual use in a script.

In this step, you start the AdminTask setTraceSpecification command in interactive mode to modify the trace level for *server1* on node *was85hostNode01* to enable maximum tracing for "com.ibm.\*" packages. You then use the administrative console to verify the success of the change.

- \_\_ 1. Start the `setTraceSpecification` command in interactive mode. In the `scriptExecutor` Console view, enter:

```
AdminTask.setTraceSpecification("-interactive")
```

A description of the command is displayed, followed by a prompt for the value of the first command parameter, *Server Name*. The name in parentheses (*serverName*) is the actual argument name that you would use to specify a value for this parameter in an option string or list.



## Information

In the scriptExecutor Console view, the cursor is not immediately positioned on the input prompt line. However, as you start typing a value, the cursor is automatically repositioned on the input prompt line.

2. In the *Server Name (serverName)* prompt, type **server1** and press **Enter**.

A prompt for the second command parameter is displayed.

3. In the *Node Name (nodeName)* prompt, type `was85hostNode01` and press **Enter**.

A prompt for the third command parameter is displayed. The asterisk (\*) in the prompt indicates that it is required.

- \_\_ 4. In the *Trace Specification (traceSpecification)* prompt, type `com.ibm.*=all` and press **Enter**.

A prompt for the fourth command parameter is displayed. This one is of type Boolean, and a default value of `false` is provided.

- \_\_ 5. In the *Persists the trace specification (persist)* prompt, press **Enter** to accept the default value of **false**.

All parameter values are now specified, and a prompt is to *Finish* (execute) or *Cancel* the command is displayed.

6. In the *Select [F, C]* prompt, press **Enter** to accept the default value of F (Finish).

The invocation syntax for the command is generated and displayed, and the command is run.

- \_\_ 7. Use the administrative console to verify that the trace level for **server1** was properly modified.

  - \_\_ a. Display the Servers view by clicking the **Servers** tab.
  - \_\_ b. Right-click **WebSphere Application Server v8.5 at localhost**, and select **Run administrative console**.
  - \_\_ c. In the Security Alert window, click **OK** to accept the security certificate and proceed.
  - \_\_ d. The administrative console opens in the Admin Console view and displays the Login page. Type **wasadmin** for **User ID**, **websphere** for **password**, and click **Log in**.
  - \_\_ e. The main page of the administrative console is shown. In the navigation pane, select **Troubleshooting > Logs and Trace**. The Logging and Tracing page is displayed in the right pane.
  - \_\_ f. In the right pane, select **server1**. The General Properties page for server1 is displayed.

- \_\_\_ g. In the General Properties page, select **Change Log Detail Levels**. The **Configuration** log detail level page is displayed.

The screenshot shows the 'Logging and tracing' interface for 'server1'. The 'Configuration' tab is selected. Under 'Change log detail levels', there is a checkbox for 'Disable logging and tracing of potentially sensitive data' and a text input field containing '\*=info', which is also highlighted with a red box. A link to 'Components and Groups' is visible below the input field.

Notice that the log level indicated on the **Configuration** tab *is not changed*. This result happens because you did not select the *persist* option in the interactive invocation of `setTraceSpecification`. Therefore, the trace setting in the configuration repository was not changed.

- \_\_\_ h. Select the **Runtime** tab. The **Runtime** log detail level page is displayed.

The screenshot shows the 'Logging and tracing' interface for 'server1'. The 'Runtime' tab is selected. Under 'Change log detail levels', there is a checkbox for 'Disable logging and tracing of potentially sensitive data' and a text input field containing '\*=info com.ibm.\*=all', which is highlighted with a red box. A link to 'Components and Groups' is visible below the input field.

The display shows that the runtime trace string was properly modified to append "com.ibm.\*=all".



## Troubleshooting

If the trace specification is not appended in the Runtime view, run `AdminTask.setTraceSpecification("-interactive")` again with the same input values.

Also, switch to the `server1` Console view and look for the log message:

```
ManagerAdmin I TRAS0018I: The trace state has changed. The  
new trace state is *=info:com.ibm.*=all.
```

- i. Click **Logout** to exit the administrative console and close the Admin Console view.

You successfully used the `AdminTask setTraceSpecification` command in interactive mode to change the trace level setting for the `server1` server.

## Section 4: Starting the command in batch mode

Starting an AdminTask command in batch mode requires that you supply all of its applicable parameters in the invocation statement by using the appropriate syntax. As illustrated in Figure 1, this task entails the following general steps:

1. Determine the target object of the command arguments, and steps.
2. Determine step arguments
3. Compose and start the command.

In the previous section, you saw how you can run an AdminTask command in interactive mode to easily discover its appropriate invocation syntax. Using this approach allows you to bypass the steps that are described. However, it requires that you run the command at least once before you can see the generated syntax.

In this section, you use these steps to construct the syntax of a `setTraceSpecification` command from scratch. The objective is to set the trace level for `server1` on node `was85hostNode01` to `"*=detail"` both in the runtime environment and the configuration repository. In the process, you discover and explore the resources available to help in performing each step.

- \_\_\_ 1. To determine the target object of a command, its arguments, and steps, use the `help()` method and the WebSphere Application Server Information Center documentation.
  - \_\_\_ a. Get help on how to use the `setTraceSpecification` command. In the `scriptExecutor` Console view, enter:

```
print AdminTask.help("setTraceSpecification")
```

Description: Set the trace specification for the server. If the server is running new trace specification takes effect immediately. This command also saves the trace specification in configuration.

Target object: None

### Arguments:

`serverName` - The name of the Server whose process definition is modified. If there is only one server in the entire configuration, then this parameter is optional.

`nodeName` - The name of the node. This is only needed for the server scopes that do not have a unique name across nodes.

\*traceSpecification - Trace Specification

**persist** - Save the trace specification to the configuration.

**Steps:**

None

The help output for the `setTraceSpecification` command indicates that it has no target object and steps, but accepts four parameters, one of which is required: `serverName`, `nodeName`, `traceSpecification` (required) and `persist`.

- \_\_ b. You can also use the Information Center to find the target object, arguments, and steps of an AdminTask command. For example, in the Information Center

browser window that you opened in the previous section, look below the description of the **setTraceSpecification** command:

### setTraceSpecification

Use the **setTraceSpecification** command to set the **trace** specification for the server. If the server is running new **trace** specification the change takes effect immediately. This command also saves the **trace** specification in configuration.

#### Target object

None

#### Required parameters

##### -serverName

Specifies the name of the server whose **trace** specification will be set. If there is only one server in the configuration, this parameter is optional. (String, required)

##### -nodeName

Specifies the node name where the server resides. If the server name is unique in the cell, this parameter is optional. (String, required)

##### -traceSpecification

Specifies the **trace** specification. (String, required)

A description of the target object for the command, its arguments, and steps is provided.



### Warning

Notice that there is an error in the Information Center documentation for the **setTraceSpecification** command, as it is missing the definition of the **persist** argument that you identified by using the **help()** method. It is a good idea to cross-check in both the Information Center and the **help()** method to make sure that you have the complete command description.

2. You have an idea of what parameters to supply with the **setTraceSpecification** command. But how do you supply them in the invocation statement? Finding an example to look at always helps. Here also, you can use the Information Center

documentation. In the browser window for the `setTraceSpecification` command documentation, scroll-down to the **Batch mode example usage section**:

### Examples

Batch mode example usage:

- Using JACL:

```
$AdminTask setTraceSpecification {-serverName server1 -nodeName node1  
-traceSpecification com.ibm.*=all=enabled}
```

- Using Jython string:

```
AdminTask.setTraceSpecification(['-serverName server1 -nodeName node1  
-traceSpecification com.ibm.*=all=enabled'])
```

- Using Jython list:

```
AdminTask.setTraceSpecification(['-serverName', 'server1', '-nodeName', 'node1',  
'-traceSpecification', 'com.ibm.*=all=enabled'])
```

Under **Using Jython list**, you see an example of the syntax of a `setTraceSpecification` command invocation that uses an options list to specify arguments.

- \_\_\_ 3. You are now ready to compose a `setTraceSpecification()` method to set the trace level to "`*=detail`". In the editor view for `ex06_setTraceLevel.py`:
- \_\_\_ a. Add statements to assign the wanted values of the `serverName`, `nodeName`, and `traceSpecification` arguments to variables.
  - \_\_\_ b. Build an options list to contain the argument name-value pairs as shown in the Information Center example that you looked at in the previous step. Also, add the argument to persist the change to the configuration. Assign the list to a variable.
  - \_\_\_ c. Start the `AdminTask.setTraceSpecification()` command, passing it the options list that you built.
  - \_\_\_ d. Finally, save the configuration change to the repository.



### Information

You must issue an `AdminConfig.save()` command because the `persist` option that you specified also changes the trace level in the configuration repository. As such, it is not committed until an explicit save is issued.

Your script should look as follows:

The screenshot shows a Jython code editor with two tabs open. The current tab is `ex06_setTraceLevel.py`, which contains the following code:

```
1 #  
2 # TODO: enter JYTHON code and save  
3 #  
4 # Assign argument values.  
5 server = "server1"  
6 node = "was85hostNode01"  
7 traceString = "*=detail"  
8 persistState = "true"  
9  
10 # Build options list.  
11 optionsList = ["-serverName", server, "-nodeName", node,  
12                 "-traceSpecification", traceString, "-persist", persistState]  
13  
14 # Invoke AdminTask command.  
15 AdminTask.setTraceSpecification(optionsList)  
16 AdminConfig.save()
```

- \_\_\_ 4. Type **Ctrl+S** to save the script.
  - \_\_\_ 5. Run the script. In the scriptExecutor Console view, enter:

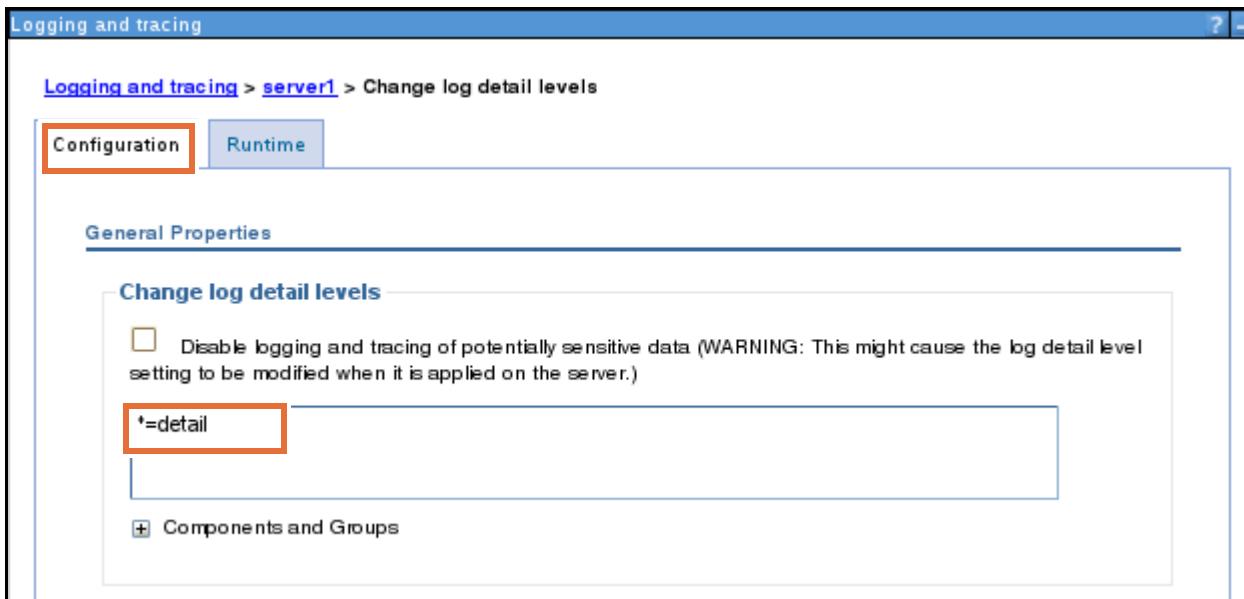
## ex06 setTraceLevel.py

The command completes, but there is not output written to the console.

- \_\_\_ 6. Use the administrative console to verify that the trace level for `server1` was properly modified.

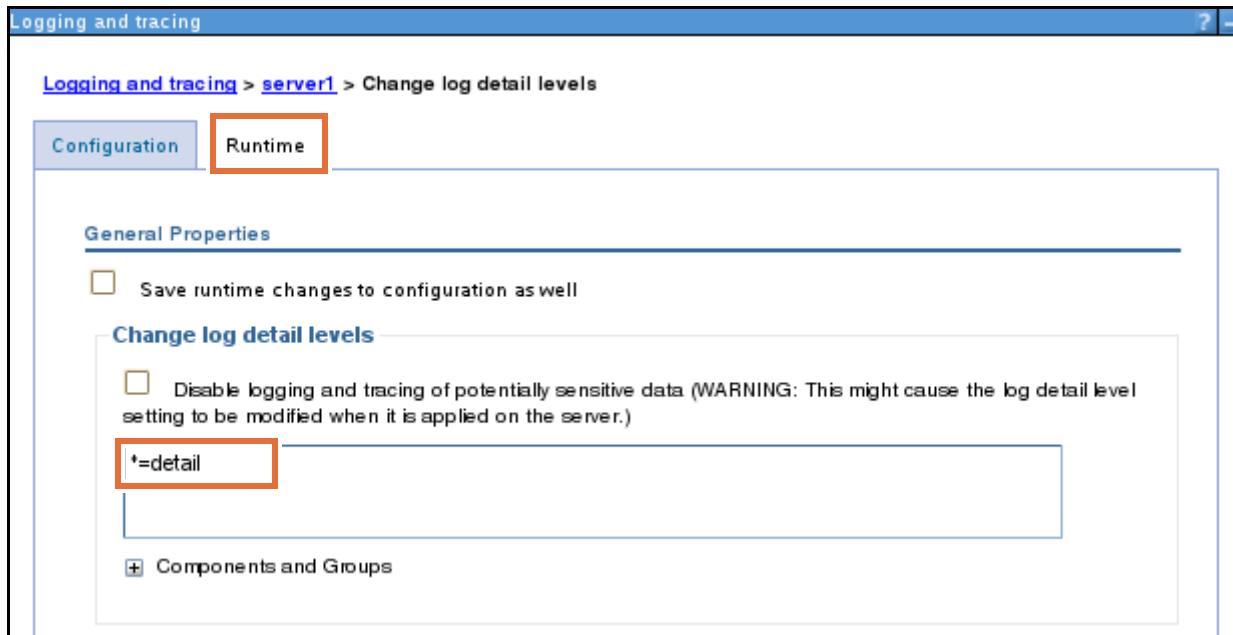
  - \_\_\_ a. Display the Servers view by clicking the **Servers** tab.
  - \_\_\_ b. Right-click **WebSphere Application Server v8.5 at localhost**, and select **Run administrative console**.
  - \_\_\_ c. The administrative console opens in the Admin Console view and displays the Login page. Type `wasadmin` for **User ID**, `websphere` for **password**, and click **Log in**.

- \_\_\_ d. The main page of the Administrative console is shown. In the navigation pane, select **Troubleshooting > Logs and Trace**. The Logging and Tracing page is displayed in the right pane.
- \_\_\_ e. In the right pane, select **server1**. The General Properties page for server1 is displayed.
- \_\_\_ f. In the General Properties page, select **Change Log Detail Levels**. The **Configuration** log detail level page is displayed.



Notice that, this time, the log level that is indicated on the **Configuration** tab was changed. This change is because you specified the `persist` option in the command invocation.

- \_\_\_ g. Select the **Runtime** tab. The **Runtime** log detail level page is displayed.



Here you see that the log level is also changed to "\*=detail" as expected.

- \_\_\_ h. Click **Logout** to exit the administrative console and close the Admin Console view.



## Troubleshooting

If the log level is not changed on the Configuration tab, check the `server1` console for the following messages:

```
ManagerAdmin I TRAS0018I: The trace state has changed. The new  
trace state is *=detail.
```

This message indicates that the trace state changed on the run time.

```
FileRepositor A ADMR0016I: User  
defaultWIMFileBasedRealm/wasadmin modified document  
cells/was85hostNode01Cell/nodes/was85hostNode01/servers/server1/  
server.xml.
```

This message indicates that trace state was changed in the configuration, that is, "persisted"

If you do not see the second message, verify that the script is explicitly setting the `persist` option to `true`.

You successfully constructed an `AdminTask` command to change the trace level setting for the `server1` server from scratch and started it in batch mode.

## ***Section 5: Working with JVM-related methods and JVM system properties***

The ServerManagement command group for the AdminTask object includes several JVM-related methods that you explore in this section. In particular,

- `setGenericJVMArguments()`
  - `setJVMIInitialHeapSize()`
  - `setJVMMaxHeapSize()`
  - `setJVMProperties()`
  - `showJVMProperties()`

In this section, you construct the syntax of a `setJVMProperties` command from scratch, with the objective of setting several JVM properties for `server1` on node `was85hostNode01` both in the runtime environment and the configuration repository. In the process, you discover and explore the resources available to help in performing each step.

- \_\_\_ 1. To determine the target object of a command, its arguments, and steps, use the `help()` method and the WebSphere Application Server Information Center documentation.
  - \_\_\_ a. Get help on how to use the `showJVMProperties` command. In the `scriptExecutor` Console view, enter:

```
print AdminTask.help("showJVMProperties")
```

Description: List Java virtual machine (JVM) configuration for the application server's process.

Target object: template or server object name.

#### Arguments:

**serverName** - The name of the Server whose process definition is modified. If there is only one server in the entire configuration, then this parameter is optional.

`nodeName` - The name of the node. This is only needed for the server scopes that do not have a unique name across nodes.

**processType** - The process type of the server. This is for zOS only.

`propertyName` - See command `setJVMProperties` for supported property names. If this parameter is not specified, then all properties are shown.

#### Steps:

None

To view all of the JVM properties of a server, do not specify the property name. Otherwise, a specific property can be viewed by using the `propertyName` argument.

- \_\_ b. To explore all of the properties of the server, you must supply only the `serverName` as an argument if that name is unique across nodes. Check the Information Center to see the format of the command.

## Examples

## Batch mode example usage:

- Using Jack:

```
$AdminTask showJVMProperties {-serverName server1 -nodeName node1 -propertyName test.property}
```

- Using Jython string:

```
AdminTask.showJVMProperties(['-serverName server1 -nodeName node1 -propertyName test.property'])
```

- Using Jython list:

```
AdminTask.showJVMProperties(['-serverName', 'server1', '-nodeName', 'node1',  
    '-propertyName', 'test.property'])
```

In the following steps, you use two batch mode formats: Jython string and Jython list.

- \_\_ c. In the scriptExecutor Console view, enter on one line:

```
print AdminTask.showJVMProperties(["-serverName",  
"server1"])
```

- \_\_ d. The output is the JVM properties of a server that can be viewed and modified in the administrative console by going to **WebSphere application servers > server1 > Java and Process Management > Process definition > Java Virtual Machine**. Notice the following properties and values.

- verboseModeGarbageCollection = false
  - initialHeapSize = 0

- maximumHeapSize = 0
- genericJvmArguments = -Xquickstart

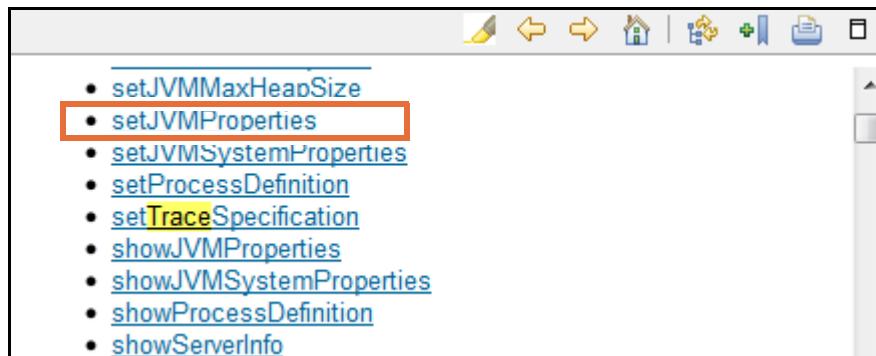
You are going to use the `setJVMProperties` method to modify these properties.



### Information

The values of zero for the `initialHeapSize` and `maximumHeapSize` mean that these properties use the default values (50 MB for `initialHeapSize` and 256 MB for `maximumHeapSize`).

- \_\_\_ 2. Check the Information Center for examples of using the `setJVMProperties` command.
  - \_\_\_ a. Go to the ServerManagement command group for the `AdminTask` object as you did previously.



\_\_\_ b. Click **setJVMproperties**.

### **setJVMProperties**

Use the **setJVMProperties** command to set the Java virtual machine (JVM) configuration for the application server.

#### **Target object**

None

#### **Required parameters**

##### **-serverName**

Specifies the name of the server for which the JVM properties will be modified. If there is only one server in the configuration, this parameter is optional. (String, required)

##### **-nodeName**

Specifies the node name where the server resides. If the server name is unique in the entire cell, this parameter is optional. (String, required)

#### **Optional parameters**

- \_\_\_ c. Scroll down to view the list of Optional parameters.

**-verboseModeGarbageCollection**

Specifies whether to use verbose debug output for garbage collection. The default is not to enable verbose garbage collection. (Boolean, optional)

**-verboseModeJNI**

Specifies whether to use verbose debug output for native method invocation. The default is not to enable verbose Java Native Interface (JNI) activity. (Boolean, optional)

**-initialHeapSize**

Specifies the initial heap size in megabytes that is available to the JVM code. (Integer, optional)

**-maximumHeapSize**

Specifies the maximum heap size available in megabytes to the JVM code. (Integer, optional)

**-runHProf**

This parameter only applies to WebSphere Application Server version. It specifies whether to use HProf profiler support. To use another profiler, specify the custom profiler settings using the hprofArguments parameter. The default is not to enable HProf profiler support. (Boolean, optional)

**-hprofArguments**

This parameter only applies to WebSphere Application Server version. It specifies command-line profiler arguments to pass to the JVM code that starts the application server process. You can specify arguments when HProf profiler support is enabled. (String, optional)

**-debugMode**

Specifies whether to run the JVM in debug mode. The default is not to enable debug mode support. (Boolean, optional)

**-debugArgs**

Specifies the command line debug arguments to pass to the JVM code that starts the application server process. You can specify arguments when the debug mode is enabled. (String, optional)

**-genericJVMArguments**

Specifies the command line arguments to pass to the JVM code that starts the application server process. (String, optional)

- \_\_\_ d. Notice the value data types of the properties that are highlighted in the screen capture. Scroll down again to view the example batch mode formats.

**Examples**

Batch mode example usage:

- Using JACL:

```
$AdminTask setJVMProperties {-serverName server1 -nodeName node1}
```

- Using Jython string:

```
AdminTask.setJVMProperties(['-serverName server1 -nodeName node1'])
```

- Using Jython list:

```
AdminTask.setJVMProperties(['-serverName', 'server1', '-nodeName', 'node1'])
```

You now have enough information to begin writing some commands to modify the JVM properties for server1.

- \_\_\_ 3. View and modify the genericJvmArguments property.
    - \_\_\_ a. First, use the showJVMProperties command to view the current value of the genericJvmArguments property. In the scriptExecutor Console view, enter on one line:

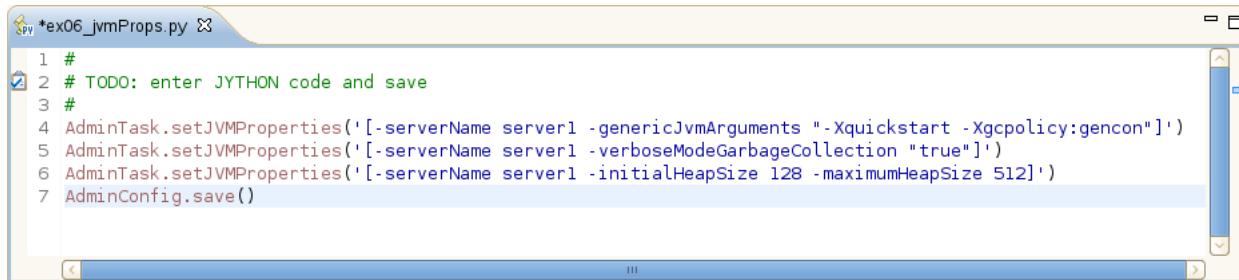
```
print AdminTask.showJVMProperties(["-serverName", "server1", "-propertyName", "genericJvmArguments"]))
```

The current value is `-Xquickstart`. You are going to append `-Xgcpolicy:gencon`. This argument sets the garbage collection policy for the server to generational-concurrent (gencon). You can use the `showJVMProperties` command to view any of the properties by specifying the name of the property.

- \_\_ b. Next, create a script file named **ex06\_jvmProps.py**. In the editor view, enter the following lines.

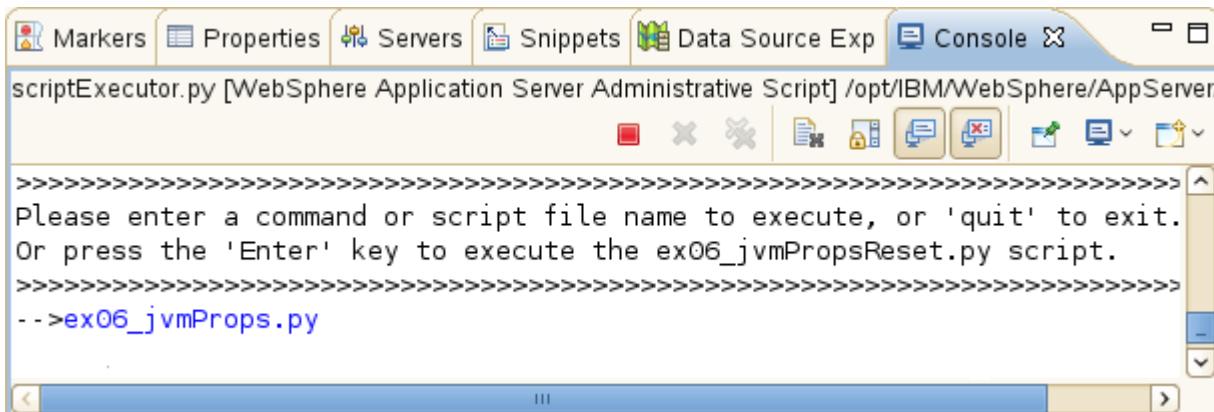
```
AdminTask.setJVMProperties('[-serverName server1  
-genericJvmArguments "-Xquickstart -XgcPolicy:gencon"]')  
  
AdminTask.setJVMProperties('[-serverName server1  
-verboseModeGarbageCollection "true"]')  
  
AdminTask.setJVMProperties('[-serverName server1 -initialHeapSize  
128 -maximumHeapSize 512]')  
  
AdminConfig.save()
```

Your script should look as follows:



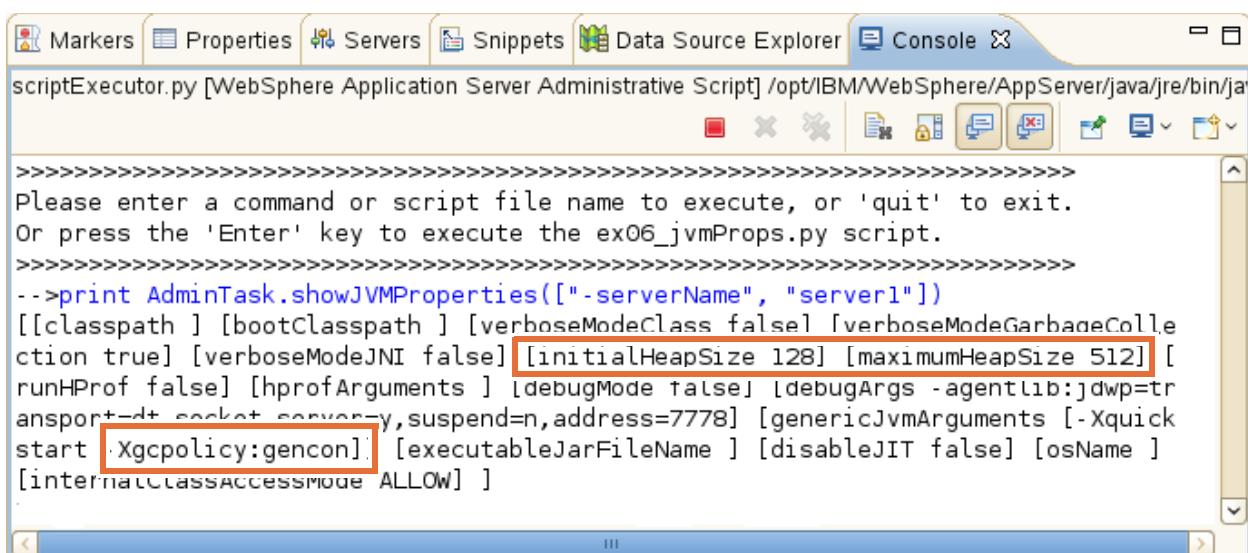
- \_\_ 4. Type **Ctrl+S** to save the script.
  - \_\_ 5. Run the script. In the scriptExecutor Console view, enter:

## ex06 jvmProps.py



- \_\_ 6. Run the showJVMProperties command to view the changes. In the scriptExecutor Console view, enter:

```
print AdminTask.showJVMProperties(["-serverName",  
"server1"])
```

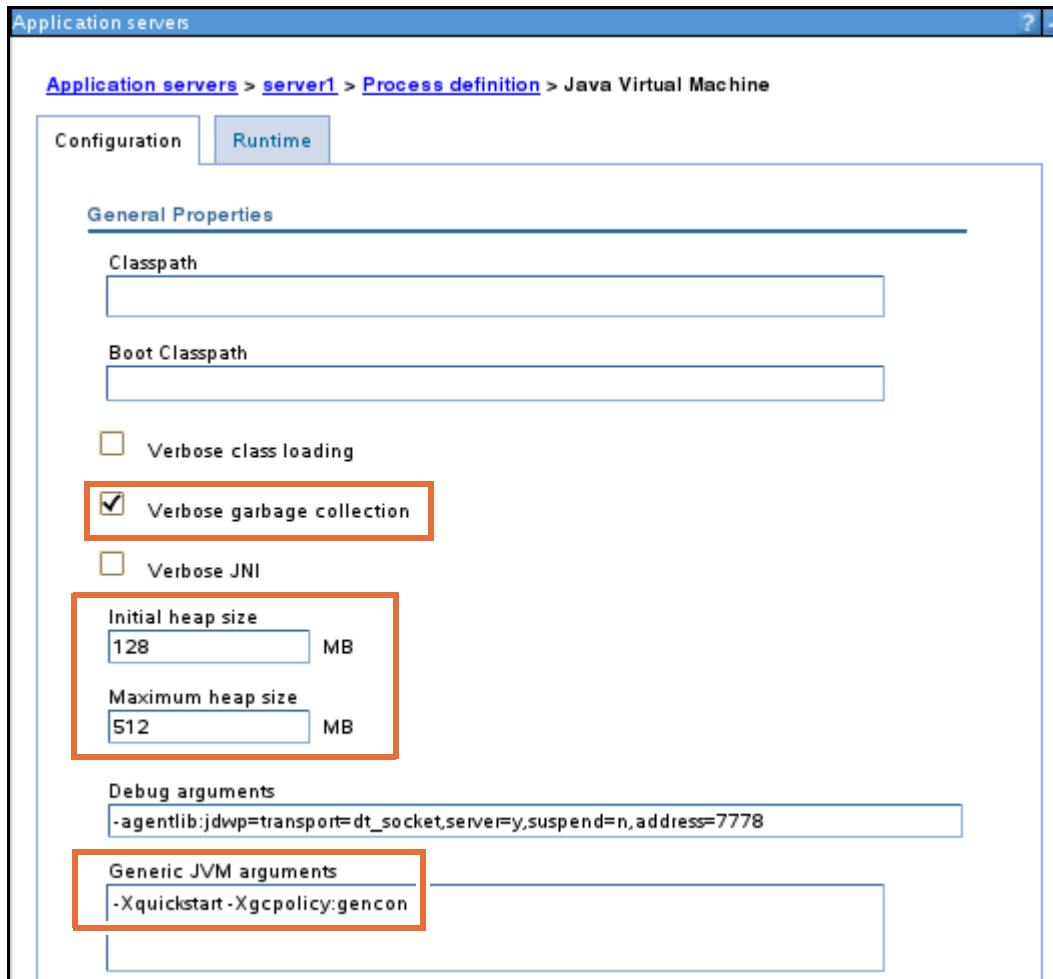


Notice that the JVM properties are modified.

- \_\_\_ 7. Use the administrative console to verify that the trace level for *server1* was properly modified.

  - \_\_\_ a. Display the Servers view by clicking the **Servers** tab.
  - \_\_\_ b. Right-click **WebSphere Application Server v8.5 at localhost**, and select **Run administrative console**.
  - \_\_\_ c. In the Security Alert window, click **OK** to accept the security certificate and proceed.
  - \_\_\_ d. The administrative console opens in the Admin Console view and displays the Login page. Type `wasadmin` for **User ID**, `websphere` for **password**, and click **Log in**.

- \_\_\_ e. The main page of the administrative console is shown. In the navigation pane, select **Servers > Server Types > WebSphere application servers > server1 > Java and Process Management > Process definition > Java Virtual Machine**.



The JVM properties are changed on the Configuration tab, but server1 must be restarted for the changes to come into effect.

- \_\_\_ 8. Log out of the administrative console.
- \_\_\_ 9. Create a script named `ex06_jvmPropsReset.py`. Copy the `ex06_jvmProps.py` script into it, and modify the lines in to change the JVM properties back to their default values.
- \_\_\_ 10. Run `ex06_jvmPropsReset.py` in the scriptExecutor Console view.
- \_\_\_ 11. Run the `showJVMProperties` command again in the scriptExecutor to verify that the JVM properties are back to their default values.

## Section 6: Cleaning up the environment

- \_\_\_ 1. Stop the scriptExecutor script. In the scriptExecutor Console window, enter: `quit`.
- \_\_\_ 2. Close any scripts that are still open in the Jython editor.
- \_\_\_ 3. Stop the server.
  - \_\_\_ a. Display the Servers view by clicking the **Servers** tab.
  - \_\_\_ b. Make sure the **WebSphere Application Server v8.5 at localhost** server is selected and click **Stop the server**.



Wait until the Servers view displays a status of "Stopped".

- \_\_\_ 4. Exit the IBM Assembly and Deploy Tools by selecting **File > Exit**. The workspace is automatically saved.

**End of exercise**

## Exercise review and wrap-up

In this exercise, you used the AdminTask object to modify the trace level for an application server process. You identified the appropriate command to use, started it in interactive mode, and then started it in batch mode. In the process, you applied for the steps how to use the AdminTask object and identified the primary methods and resources that are required to successfully complete each step. You developed and ran the resulting Jython script in the IBM Assembly and Deploy Tools.



# Exercise 7. Creating and configuring the Plants server environment with scripting

## What this exercise is about

In this exercise, you create and configure the server environment for the PlantsByWebSphere application. Using response files, you create the necessary deployment manager and managed node profiles. You then develop Jython administrative scripts to create the cluster and clusters members that comprise the Plants server environment.

## What you should be able to do

At the end of this exercise, you should be able to:

- Create a deployment manager and a managed node profile by using a response file
- Develop Jython scripts to create a cluster and add cluster members
- Develop Jython scripts to query the state of a cluster, and start or stop a cluster

## Introduction

The deployment environment for the PlantsByWebSphere application is based on a distributed server topology and consists of:

- A deployment manager node
- Two managed nodes that each contain an PlantsByWebSphere application server
- A cluster that contains the two application servers as cluster members

Figure 1 highlights the servers and server resources that you create and configure in the next several exercises.

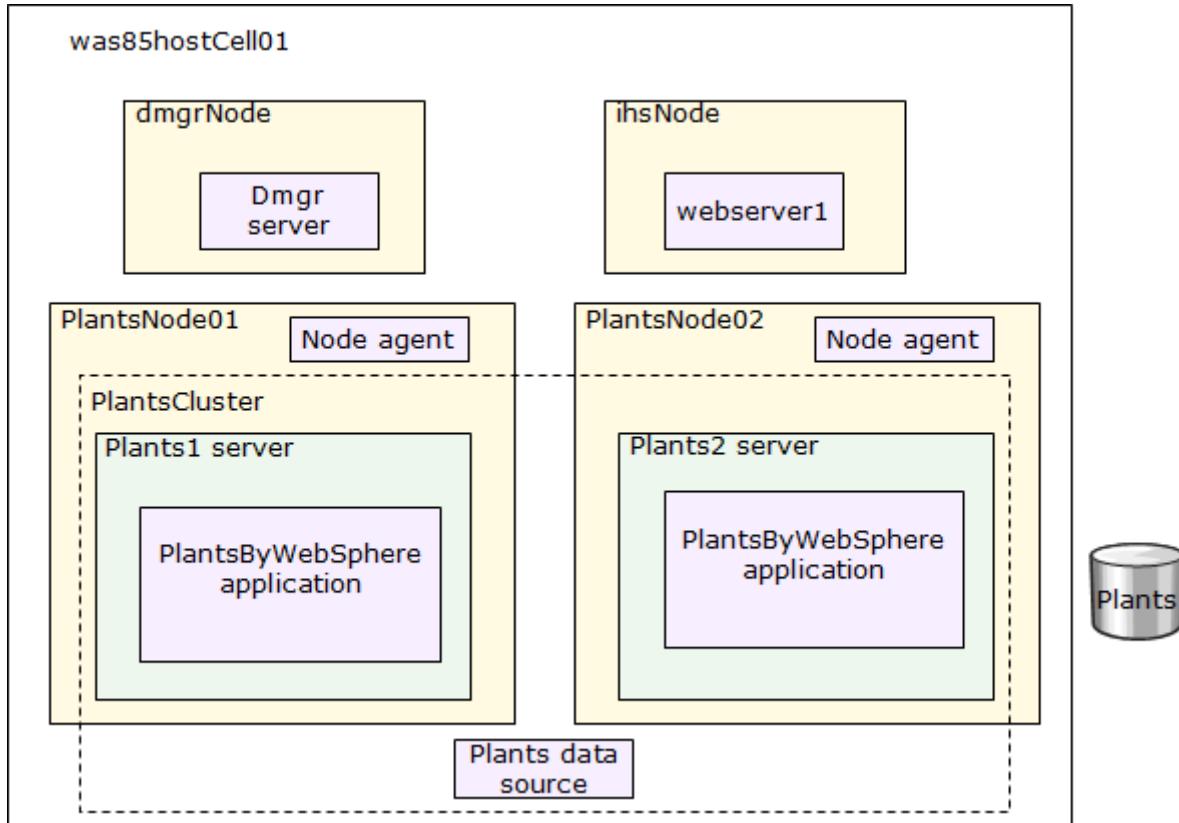


Figure 1 - Plants deployment environment servers and server resources

The specific steps that you follow to create the Plants server environment in this exercise are outlined.

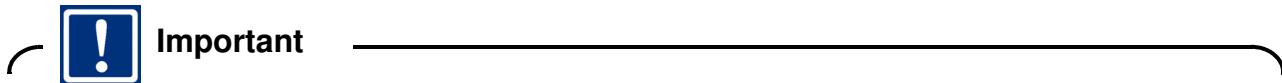
1. Create the deployment manager and managed node profiles in silent mode.
2. Prepare the script development environment in the IADT.
3. Create the Plants cluster by using scripting
4. Create the Plants cluster members by using scripting.
5. Develop scripts to query the state of a cluster, and start or stop a cluster.

## Requirements

To complete this exercise, you must have the WebSphere Application Server Network Deployment V8.5 product installed. You also require the IBM Assembly and Deploy Tools for WebSphere Administration V8.5.



## Exercise instructions



### Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

`<was_root>`: /opt/IBM/WebSphere/AppServer

`<profile_root>`: /opt/IBM/WebSphere/AppServer/profiles

### Section 1: Creating the deployment manager and managed node profiles

WebSphere Application Server provides two ways to create a profile: the *Profile Management Tool* and the *manageprofiles* command. The former uses a graphical user interface to collect the properties of the profile to create, while the latter collects them from the command line or from a provided response file.

In this section, you create a deployment manager profile (*dmgr*) and two managed node profiles (*PlantsProfile1* and *PlantsProfile2*) with response files and the *manageprofiles* command. This command creates the *dmgrNode*, *PlantsNode01*, *PlantsNode02* nodes, and their respective servers as illustrated in Figure 1.

For each type of profile, you first discover how to build the required response file, that is, what properties are required. Then, you create a response file with the wanted Plants properties and start the *manageprofiles* command to process it. Finally, you verify the successful profile creation by using the administrative console.

- \_\_\_ 1. Start by getting help about how to use the *manageprofiles* command.
  - \_\_\_ a. Open a command prompt window and go to the `<was_root>/bin` directory.
  - \_\_\_ b. In the command prompt, enter the following command:

```
./manageprofiles.sh -help
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/bin # ./manageprofiles.sh -help

Function:
Creates, lists, alters or deletes profiles

Syntax:
manageprofiles -<mode> -<argument> <argument parameter> ...

The available modes are:
create
augment
delete
unaugment
unaugmentAll
deleteAll
listProfiles
listAugments
backupProfile
restoreProfile
getName
getPath
validateRegistry
validateAndUpdateRegistry
getDefaultName
setDefaultName
response
help

Note: Command-line arguments are case sensitive.
Note: If argument accepts a parameter containing spaces, the parameter must be enclosed in "double quotes".
Note: The default profile template is "default" and may be overridden by the -templatePath switch.
Note: Mode specific arguments are described in the detailed help for each mode.
      To see the detailed help on each mode enter: -<mode> -help
was85host:/opt/IBM/WebSphere/AppServer/bin #
```

The output indicates that detailed help can further be retrieved by specifying the *mode* in which you want to use the *manageprofiles* command. The two modes that are of interest to you are *create* and *response*, since you want to create a profile by using a response file.

- c. Get specific help on the **create** mode. In the command prompt, enter:

```
./manageprofiles.sh -create -help
```

The screenshot shows a terminal window titled "Terminal". The command `./manageprofiles.sh -create -help` is entered at the prompt. The output is as follows:

```
File Edit View Terminal Tabs Help  
was85host:/opt/IBM/WebSphere/AppServer/bin # ./manageprofiles.sh -create -help  
  
Function:  
Creates a new profile  
  
Syntax:  
manageprofiles -create -<argument> <argument parameter> ...  
  
Arguments:  
The following command line arguments are required for this mode:  
-templatePath <argument parameter>; The fully qualified path name of the profile template that is located on the file system.  
-profileName <argument parameter>; The name of the profile.  
-profilePath <argument parameter>; The intended location of the profile in the file system.  
  
The following command line arguments are optional, and have no default values:  
-isDefault <argument parameter>; Make this profile the default target of command that do not use their profile parameter.  
-omitAction <argument parameter>; Omit optional features.  
  
Note: Command-line arguments are case sensitive.  
Note: If argument accepts a parameter containing spaces, the parameter must be enclosed in "double quotes".  
Note: The default profile template is "default" and may be overridden by the -templatePath switch.  
Note: Each profile template will have its own set of required and optional arguments.  
was85host:/opt/IBM/WebSphere/AppServer/bin #
```

This time the output is more verbose. It shows the name and description of the command-line arguments that are required or optional when creating a profile. These arguments are what you put in a response file if you do not want to specify them on the command line. Notice the two required arguments named **-create** and **-templatePath**. The output indicates that more specific help is available based on the template that you want to use to create the profile.



## Information

Profiles are created based on templates that are supplied with the product. These templates are in `<was_root>/profileTemplates`. Each template consists of a set of files that provide the initial settings for the profile and a list of actions to perform after the profile is created. Currently, you cannot modify these templates or create new ones. When you create a profile by using `manageprofiles`, you must specify one of the following templates:

- **default**: Used for application server profiles
- **management**: Used for deployment manager profiles (`dmgr` is deprecated)
- **managed**: Used for custom profiles
- **cell**: used for cell profiles

- \_\_\_ d. The first profile that you want to create is a deployment manager profile. Therefore, in the command prompt, enter the following command on one line to obtain more specific help on how to create one:

```
./manageprofiles.sh -create -templatePath  
/opt/IBM/WebSphere/AppServer/profileTemplates/dmgr -help
```

The screenshot shows a terminal window titled "Terminal". The command entered was `./manageprofiles.sh -create -templatePath /opt/IBM/WebSphere/AppServer/profileTemplates/dmgr -help`. The output provides help for creating a new profile, including syntax and required arguments.

```
was85host:/opt/IBM/WebSphere/AppServer/bin # ./manageprofiles.sh -create -templatePath /opt/IBM/WebSphere/AppServer/profileTemplates/dmgr -help

Function:  
Creates a new profile

Syntax:  
manageprofiles -create -<argument> <argument parameter> ...

Arguments:  
The following command line arguments are required for this mode:  
-templatePath <argument parameter>: The fully qualified path name of the profile template that is located on the file system.
```

The following command line arguments are required for this mode, but are defaulted if no values are supplied:

- profileName: The name of the profile.
- profilePath: The intended location of the profile in the file system.
- hostName: The domain name system (DNS) host name or IP address of this computer.

If using IPv6, then specify the IP address.

- nodeName: The node name of the profile. The name must be unique within its cell.
- cellName: The cell name of the profile. The cell name must be unique for each profile.
- enableAdminSecurity: Specify true to enable administrative security for the profile that is to be created.
- serverType: Specifies the type of management server to create. This can be one of the following: DEPLOYMENT\_MANAGER, JOB\_MANAGER, or ADMIN\_AGENT.
- personalCertDN: The DN for the personal certificate to be created. Note: This cannot be specified along with any of the parameters used for importing a personal certificate.
- personalCertValidityPeriod: The validity period of the personal certificate to be created. Note: This cannot be specified along with any of the parameters used for importing a personal certificate.
- signingCertDN: The DN for the signing certificate to be created. Note: This cannot be specified along with any of the parameters used for importing a signing certificate.
- signingCertValidityPeriod: The validity period of the signing certificate to be created. Note: This cannot be specified along with any of the parameters used for importing a signing certificate.
- keyStorePassword: The keystore password for the signing certificate to be created. Note: This cannot be specified along with any of the parameters used for importing a signing certificate.
- useSAFSecurity: Enables SAF security when used in conjunction with -enableAdminSecurity parameter. Only valid on the os390 platform.

The following command line arguments are optional, and have no default values:

- isDefault <argument parameter>: Make this profile the default target of commands that do not use their profile parameter.
- omitAction <argument parameter>: Omit optional features.
- importPersonalCertKS <argument parameter>: The path to the keystore for the personal certificate to be imported. Note: This cannot be specified along with any of the parameters used for creating a personal certificate.
- importPersonalCertKSType <argument parameter>: The keystore type of the personal certificate to be imported. Note: This cannot be specified along with any of the parameters used for creating a personal certificate.
- importPersonalCertKSPassword <argument parameter>: The keystore password of the personal certificate to be imported. Note: This cannot be specified along with any of the parameters used for creating a personal certificate.
- importPersonalCertKSAlias <argument parameter>: The keystore alias of the personal certificate to be imported. Note: This cannot be specified along with any of the parameters used for creating a personal certificate.
- importSigningCertKS <argument parameter>: The path to the keystore for the signing certificate to be imported. Note: This cannot be specified along with any of the parameters used for creating a signing certificate.
- importSigningCertKSType <argument parameter>: The keystore type of the signing certificate to be imported. Note: This cannot be specified along with any of the parameters used for creating a signing certificate.
- importSigningCertKSPassword <argument parameter>: The keystore password of the signing certificate to be imported. Note: This cannot be specified along with any of the parameters used for creating a signing certificate.
- importSigningCertKSAlias <argument parameter>: The keystore alias of the signing certificate to be imported. Note: This cannot be specified along with any of the parameters used for creating a signing certificate.

```

    -adminPassword <argument parameter>: Specify the password for the name specified
with the adminUserName parameter
    -serviceUserName <argument parameter>: Specify the name of the user you wish this
service to be run as.
    -portsFile <argument parameter>: An optional parameter that specifies the path to
a file that defines port settings for the new profile. Do not use this parameter with the
-startingPort or -defaultPorts parameters.
    -startingPort <argument parameter>: Specify the starting port number for generati
ng all ports for the profile. Do not use this parameter with the -portsFile or -defaultPor
ts parameters.
    -defaultPorts <argument parameter>: Accept the default ports for the new profile.
Do not use this parameter with the -portsFile or -startingPort parameters.
    -validatePorts <argument parameter>: Specify the ports should be validated to ens
ure they are not reserved or in use.

```

Note: Command-line arguments are case sensitive.

You can now see all of the arguments that are required, optional with default values, or optional without default values for creating a deployment manager profile.

- \_\_\_ e. Now that you have an idea of what to specify when creating a deployment manager profile by using *manageprofiles*, how do you specify it in a response file? To help answer this question, refer to the Information Center.
- \_\_\_ f. Do a search on “*manageprofiles*”. Click the match named **manageprofiles command** that is associated with the Network Deployment (Distributed operating systems), Version 8.5. Scroll-down to the section that describes the **-response** argument. An example response file is provided.

#### **-response reponse\_file**

Accesses all API functions from the command line using the **manageprofiles** command.

The command line interface can be driven by a response file that contains the input arguments for a given command in the properties file in key and value format. To determine which input arguments are required for the various types of profile templates and action, use the **manageprofiles** command with the **-help** parameter.

Use the following example response file to run a create operation:

```

create
profileName=testResponseFileCreate
profilePath=profile_root
templatePath=app_server_root/profileTemplates/default
nodeName=myHostName
cellName=myCellName
hostName=myHostName
omitAction=myOptionalAction1,myOptionalAction2

```

Note the format of the file that is illustrated in the example:

- The first line contains the profile action to perform (*create*).
- The remaining lines consist of “name=value” pairs where “name” is the name of a valid argument.

- The deployment manager profile template (`dmgr`) is deprecated and replaced by the management profile template with a `DEPLOYMENT_MANAGER` serverType.
2. You are now ready to build a response file that creates a deployment manager profile. Using a text editor (for example, gedit), create a response file named `createDmgrResp.txt` in the `/usr/Software/Scripts/Exercise07` folder. The specific profile creation argument values that are required are provided in the table.

**Table 1: Argument values for deployment manager profile creation**

	<b>Argument name</b>	<b>Argument value</b>
1	<code>templatePath</code>	<code>&lt;was_root&gt;/profileTemplates/management</code>
2	<code>serverType</code>	<code>DEPLOYMENT_MANAGER</code>
3	<code>profileName</code>	<code>Dmgr</code>
4	<code>profilePath</code>	<code>&lt;profile_root&gt;/Dmgr</code>
5	<code>nodeName</code>	<code>dmgrNode</code>
6	<code>enableAdminSecurity</code>	<code>true</code>
7	<code>adminUserName</code>	<code>wasadmin</code>
8	<code>adminPassword</code>	<code>websphere</code>

**Important**

Remember to specify the wanted profile action in the first line (create).

3. Verify your response file. It should look as follows:

```
create
templatePath=/opt/IBM/WebSphere/AppServer/profileTemplates/management
serverType=DEPLOYMENT_MANAGER
profileName=Dmgr
profilePath=/opt/IBM/WebSphere/AppServer/profiles/Dmgr
nodeName=dmgrNode
enableAdminSecurity=true
adminUserName=wasadmin
adminPassword=websphere
```

4. Start `manageprofiles` to create the `dmgr` deployment profile.
- a. In a terminal window, go to `<was_root>/bin`

- \_\_\_ b. Enter the following command on one line:

```
./manageprofiles.sh -response
/usr/Software/Scripts/Exercise07/createDmgrResp.txt
```

As the command runs, use the *CPU Usage* indicator in the GNOME System Monitor to monitor activity.



After a while, the message `INSTCONFSUCCESS: Success: Profile Dmgr now exists` is displayed indicating successful execution.

If you do not see the `INSTCONFSUCCESS` message or get an error message, there is more than likely an error in your response file. To troubleshoot it further, look at the created log file and follow the recovery instructions that are described in the Troubleshooting box.

## Troubleshooting

The `manageprofiles` command creates a log file in the `<was_root>/logs/manageprofiles` directory named `Profile_name_create.log` each time it is started. This file contains trace messages for all events that occur during the creation of the named profile. It shows the final status of the process by using one of the following return codes:

- `INSTCONFFAIL` = Total profile creation failure
- `INSTCONFSUCCESS` = Successful profile creation
- `INSTCONFPARTIALSUCCESS` = Profile creation errors occurred but the profile is still functional. More information identifies the errors.

If you do not get an `INSTCONFSUCCESS` return code, look in the log for any message that might give you an indication of the error. To recover:

- \_\_\_ 1. Review your response file to make sure that you specified the wanted arguments and values correctly.
- \_\_\_ 2. If the return code is `INSTCONFPARTIALSUCCESS`, delete the partially created profile by issuing the following commands in the command prompt window:

```
cd <was_root>/bin
./manageprofiles.sh -delete -profileName Dmgr
```

- \_\_\_ 3. If the profile root folder exists (<was\_root>/profiles/Dmgr), delete it.
- \_\_\_ 4. Start *manageprofiles* again with the correct response file.

- \_\_\_ 5. Next, start the deployment manager in preparation for the creation of the two managed node profiles.



### Important

The deployment manager process must be started before the creation of the remaining profiles. When the *PlantsProfile1* and *PlantsProfile2* profiles are created, they are federated to the running deployment manager cell. If the deployment manager process is not running, federation fails and the managed node profiles are not created successfully.

- \_\_\_ a. Using the terminal window, go to the <profile\_root>/Dmgr/bin directory.
- \_\_\_ b. Enter the command:

```
./startManager.sh
```

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./startManager.sh
ADMU0116I: Tool information is being logged in file
          /opt/IBM/WebSphere/AppServer/profiles/Dmgr/logs/dmgr/startServer.log
ADMU0128I: Starting tool with the Dmgr profile
ADMU3100I: Reading configuration for server: dmgr
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server dmgr open for e-business; process id is 1135
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin #
```

- \_\_\_ c. After a few moments, the dmgr server process starts. Wait until you see the message Server dmgr open for e-business; process id is \_\_\_\_\_. Copy your process id on the blank line for future reference.
- \_\_\_ 6. Now, build a response file to create the *PlantsProfile1* managed node profile.
- \_\_\_ a. Using the same approach that you used in the previous step, start by getting help on the different types of arguments available when creating a profile by using a *managed* profile template. In the command prompt, enter:

```
./manageprofiles.sh -create -templatePath
/opt/IBM/WebSphere/AppServer/profileTemplates/managed -help
```

The screenshot shows a terminal window with the title "Terminal". The window contains the command `./manageprofiles.sh -create -templatePath /opt/IBM/WebSphere/AppServer/profileTemplates/managed -help`. The output of the command is displayed in red text at the top of the terminal window.

The help output lists and describes all of the arguments that are required, optional with default values, or optional without default values for creating a managed node profile.

- \_\_\_ b. Using a text editor (for example, gedit, Emacs, or vi), create a response file named **createPlantsProfile1Resp.txt** in the `/usr/Software/Scripts/Exercise07` folder. The specific profile creation argument values that are required are provided in the table.

**Table 2: Argument values for PlantsProfile1 profile creation**

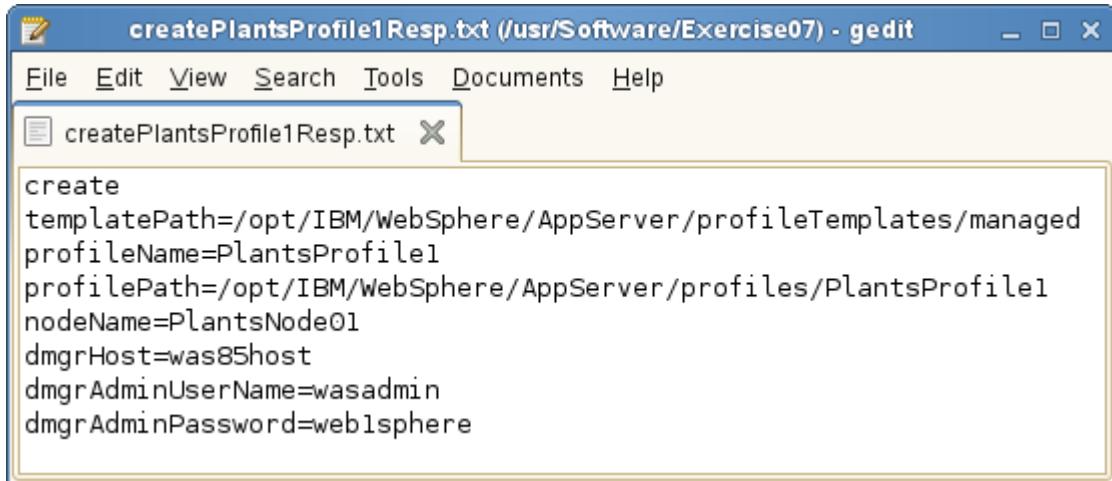
	Argument name	Argument value
1	templatePath	<was_root>/profileTemplates/managed
2	profileName	PlantsProfile1
3	profilePath	<profile_root>/PlantsProfile1
4	nodeName	PlantsNode01
5	dmgrHost	was85host
6	dmgrAdminUserName	wasadmin
7	dmgrAdminPassword	web1sphere



### Important

Remember to specify the wanted profile action in the first line (`create`).

- \_\_\_ c. Verify your response file. It should look as follows:



```
create
templatePath=/opt/IBM/WebSphere/AppServer/profileTemplates/managed
profileName=PlantsProfile1
profilePath=/opt/IBM/WebSphere/AppServer/profiles/PlantsProfile1
nodeName=PlantsNode01
dmgrHost=was85host
dmgrAdminUserName=wasadmin
dmgrAdminPassword=websphere
```

- \_\_\_ 7. Start *manageprofiles* to create the *PlantsProfile1* managed node profile.

- \_\_\_ a. In a terminal window, go to <was\_root>/bin  
 \_\_\_ b. Enter the following command on one line:

```
./manageprofiles.sh -response
/usr/Software/Scripts/Exercise07/createPlantsProfile1Resp.txt
```



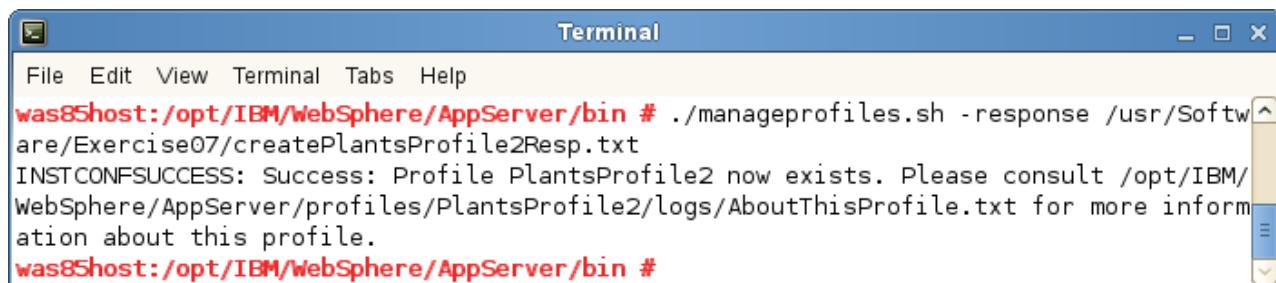
```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/bin # ./manageprofiles.sh -response /usr/Software/Scripts/Exercise07/createPlantsProfile1Resp.txt
INSTCONFSUCCESS: Success: Profile PlantsProfile1 now exists. Please consult /opt/IBM/WebSphere/AppServer/profiles/PlantsProfile1/logs/AboutThisProfile.txt for more information about this profile.
was85host:/opt/IBM/WebSphere/AppServer/bin #
```

As the command runs, use the *CPU Usage* indicator in the GNOME System Monitor to monitor activity. After a few minutes, the message `INSTCONFSUCCESS: Success: Profile PlantsProfile1 now exists` is displayed indicating successful execution.

If you do not see the `INSTCONFSUCCESS` message or get an error message, there is more than likely an error in your response file. To troubleshoot it further, look at the created log file and follow the recovery instructions that are described in the Troubleshoot box in step 4.

- \_\_\_ 8. The last profile to create is *PlantsProfile2*. Like *PlantsProfile1*, it is a managed node profile. Since you just built a response file for one, use the provided solutions response file to create *PlantsProfile2*.
- \_\_\_ a. Copy `/usr/Solutions/Exercise07/createPlantsProfile2Resp.txt` to `/usr/Software/Scripts/Exercise07`.
- \_\_\_ 9. Start *manageprofiles* to create the *PlantsProfile2* managed node profile.
- \_\_\_ a. In the terminal window, enter:

```
./manageprofiles -response  
/usr/Software/Scripts/Exercise07/createPlantsProfile2Resp.txt
```



A screenshot of a terminal window titled "Terminal". The window has a blue header bar with the title and standard window controls. The main area contains a command-line interface. The user has run the command `./manageprofiles -response /usr/Software/Scripts/Exercise07/createPlantsProfile2Resp.txt`. The terminal output shows the command prompt `was85host:/opt/IBM/WebSphere/AppServer/bin #`, followed by the message `INSTCONFSUCCESS: Success: Profile PlantsProfile2 now exists. Please consult /opt/IBM/WebSphere/AppServer/profiles/PlantsProfile2/logs/AboutThisProfile.txt for more information about this profile.`. The terminal window has scroll bars on the right side.

As the command runs, use the *CPU Usage* indicator in the GNOME System Monitor to monitor activity. After 3 to 5 minutes, the message `INSTCONFSUCCESS: Success: Profile PlantsProfile2 now exists` is displayed indicating successful execution.

If you do not see the `INSTCONFSUCCESS` message or get an error message, there is more than likely an error in your response file. Make sure that you specified the correct host name and try again.

- 10. Finally, used the administrative console to verify that the wanted cell and node configurations were created as a result of creating these three profiles.
  - a. Start the Firefox web browser, and enter the web address  
`http://was85host:9061/ibm/console`

- \_\_ b. In the Untrusted Connection window, click **I Understand the Risks**.



## This Connection is Untrusted

You have asked Firefox to connect securely to **was85host:9043**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

### What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

**Get me out of here!**

#### ► Technical Details

#### ▼ I Understand the Risks

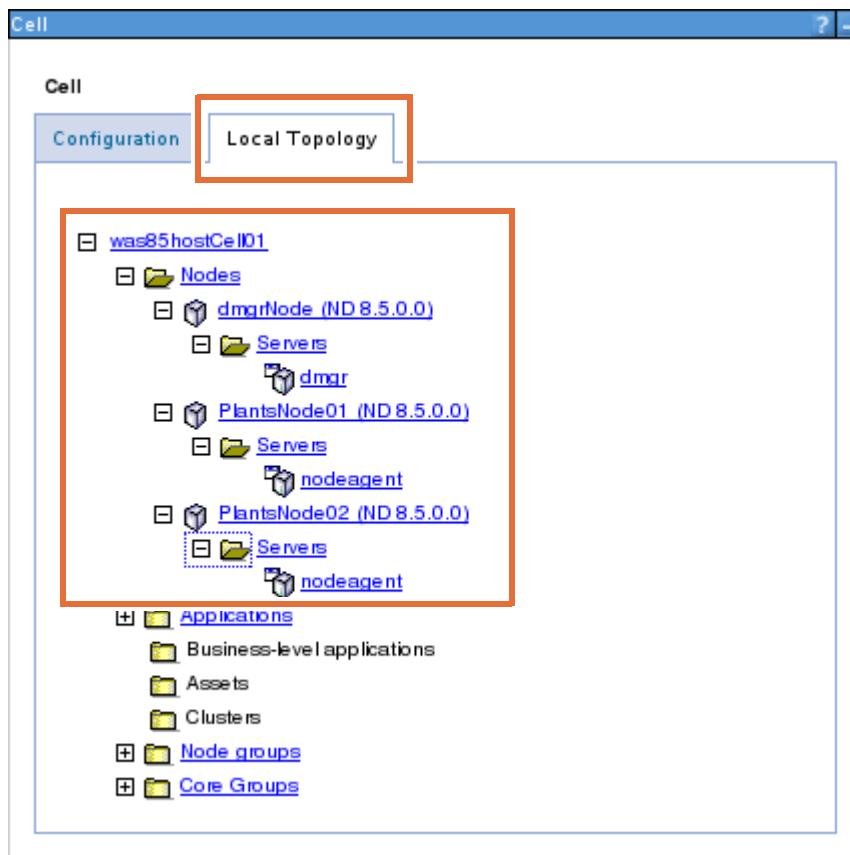
If you understand what's going on, you can tell Firefox to start trusting this site's identification. **Even if you trust the site, this error could mean that someone is tampering with your connection.**

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

**Add Exception...**

- \_\_ c. Click **Add Exception**.
- \_\_ d. Click **Confirm Security Exception**.
- \_\_ e. The administrative console opens and displays the Login page. Type `wasadmin` for User ID, `web1sphere` for password, and click **Log in**.
- \_\_ f. The main page of the Administrative console is displayed. In the navigation pane, select **System administration > Cell**.
- \_\_ g. In the right pane, click the **Local Topology** tab and expand **was85hostCell01 > nodes**. The topology tree shows the `dmgrNode`, `PlantsNode01`, and `PlantsNode02` nodes.

- \_\_\_ h. Expand each node to display the servers they contain.



Notice that *dmgrNode* contains the *dmgr* server, while *PlantsNode01* and *PlantsNode02* contain only a *nodeagent* server. No application servers exist yet.

- \_\_\_ i. Log out of the administrative console and close the browser window.

You successfully created a deployment manager and managed node profiles in silent mode by using the *manageprofiles* command.

## Section 2: Preparing the script development environment in the IADT

Before you start writing the scripts to create the server resources that the PlantsByWebSphere application requires, prepare your development environment in the IBM Assembly and Deploy Tools.

In addition to the *scriptExecutor.py* script, several Jython utility scripts and helper files are provided to further facilitate script development in the IADT and illustrate good script development practices. They consist of:

- *setScriptEnvironment.py*: A utility script to set global environment variables and append workspace paths to the interpreter system path
- *PropertyLoaderClass.py*: A Jython utility class that is used to retrieve Plants configuration properties from an external property file (*plants.properties*)
- *plants.properties*: A property file that contains the constant values of configuration properties for Plants (for example, deployment manager node name, cluster name, web server name)
- *ex07\_<scriptName>.py*: A skeleton file to be used as a starting point for developing a **function** that is associated with the script identified by *<scriptName>*
- *ut07\_<scriptName>.py*: A skeleton file to be used as starting for developing the **mainline** (unit test) code to start the corresponding function of the script that *<scriptName>* identifies.

These utility and helper scripts are described in more detail later in the section.

In this section, you open a new workspace in the IBM Assembly and Deploy Tools (IADT) and create a Jython project to hold the scripts that you develop. You also import the utility and helper scripts that are identified, and import and run the *scriptExecutor*. Finally, you create a server definition for *dmgr*, the deployment manager server, so that you can control the server directly from inside the tool.

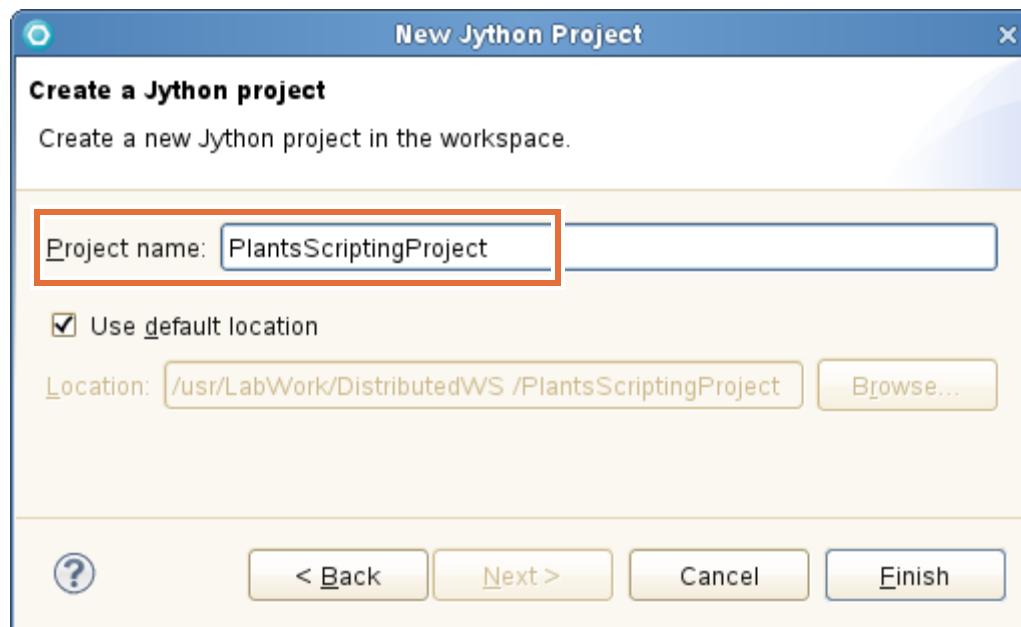
- \_\_\_ 1. Start the IBM Assembly and Deploy Tools and open a new workspace.
  - \_\_\_ a. On the desktop, double-click the **IBM Assembly and Deploy Tools** icon. If there is no IADT icon, open a terminal window, go to `/opt/IBM/SDP`, and enter the command `./eclipse &`
  - \_\_\_ b. In the Workspace Launcher window, type `/usr/LabWork/DistributedWS` in the Workspace field.
  - \_\_\_ c. Click **OK**. The IADT opens and displays the Java EE perspective by default.

\_\_ 2. Create a Jython project called **PlantsScriptingProject**.

\_\_ a. From the main menu, select **File > New > Project**.

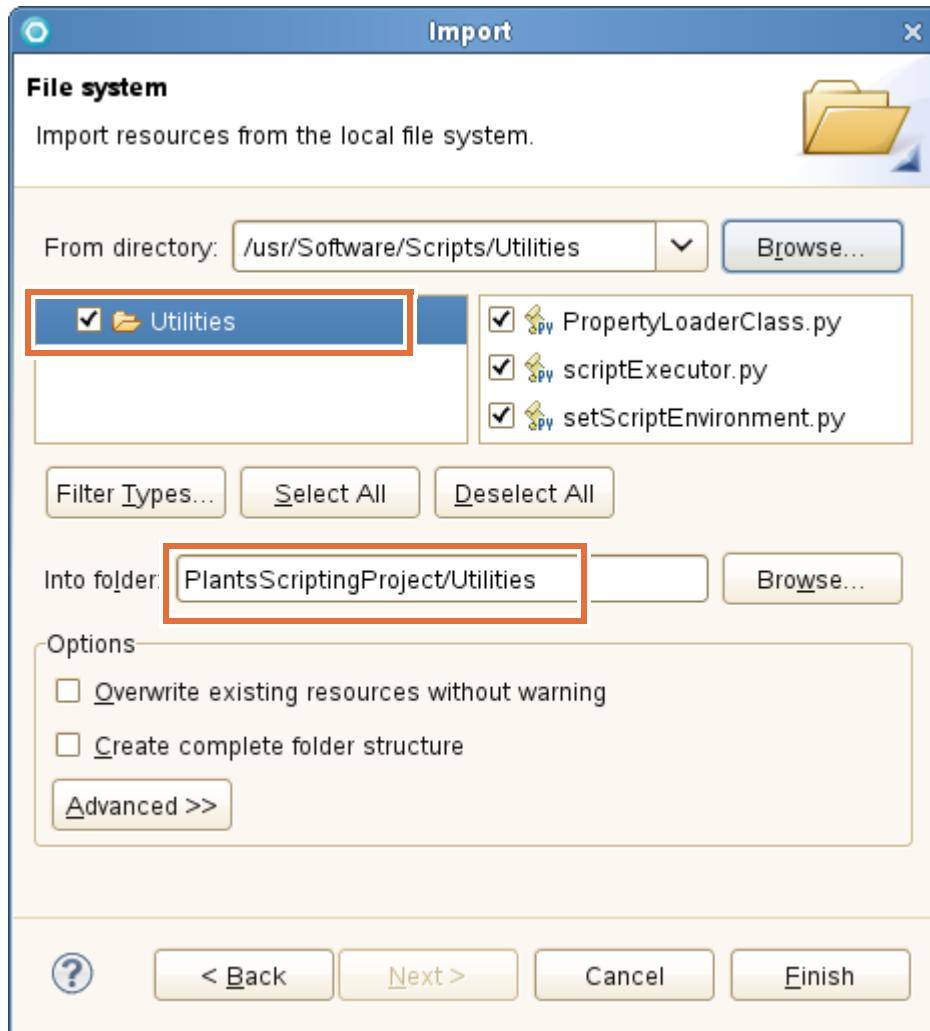


- \_\_ b. In the *Select a wizard* dialog, scroll down, expand **Jython**, and select **Jython Project**.
- \_\_ c. Click **Next**.
- \_\_ d. In the *Create a Jython project* dialog, type **PlantsScriptingProject** in the Project name field.



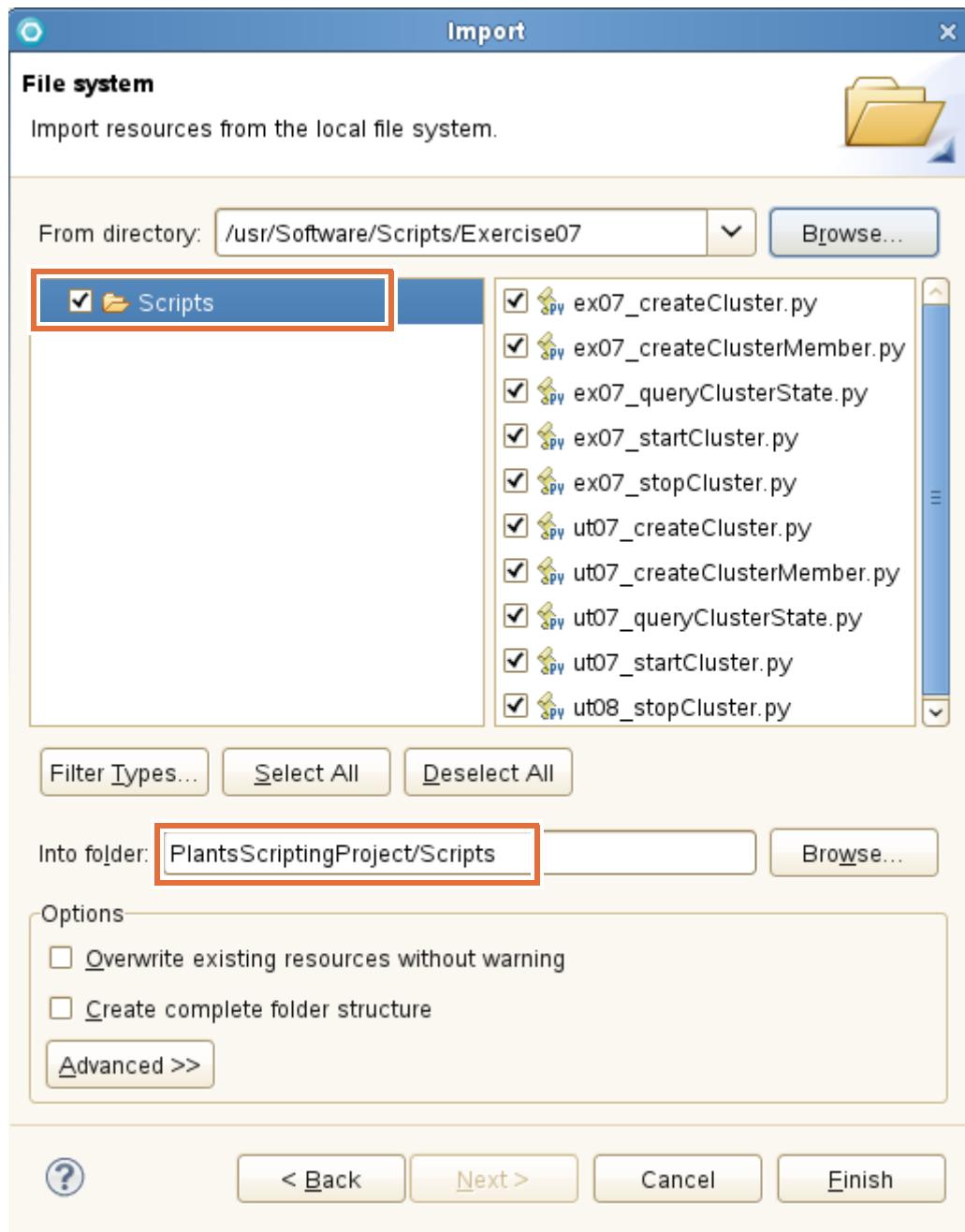
- \_\_ e. Click **Finish**. The PlantsScriptingProject is displayed in the Enterprise Explorer view.

3. Import the **PropertyLoaderClass.py**, **scriptExecutor.py**, and **setScriptEnvironment.py** utility scripts.
- In the Enterprise Explorer view, right-click **PlantsScriptingProject** and select **Import**.
  - In the *Import select* dialog, expand **General** and select **File system**.
  - Click **Next**.
  - In the *File system* dialog, click **Browse** next to the From directory field.
  - In the *Import from directory* dialog, go to **/usr/Software/Scripts**, select **Utilities**, and click **OK**.
  - In the *File system* dialog:
    - Select the **Utilities** check box to select all of the folder contents.
    - Type **/Utilities** at the end of **PlantsScriptingProject** in the Into folder field. This action results in the files that are imported in a folder named *Utilities* under the project.



- Click **Finish**.
- \_\_\_ g. In the Enterprise Explorer view, expand **PlantsScriptingProject > Utilities**. The **PropertyLoaderClass.py**, **scriptExecutor.py**, and **setScriptEnvironment.py** scripts are displayed in the folder.
- \_\_\_ 4. Import the skeleton function and mainline scripts that are designed to help you in your script development.
- \_\_\_ a. In the Enterprise Explorer view, right-click **PlantsScriptingProject** and select **Import**.
  - \_\_\_ b. In the *Import select* dialog, expand **General** and select **File system**.
  - \_\_\_ c. Click **Next**.
  - \_\_\_ d. In the *File system* dialog, click **Browse** next to the From directory field.
  - \_\_\_ e. In the *Import from directory* dialog, go to **/usr/Software/Scripts**, select **Exercise07**, and click **OK**.
  - \_\_\_ f. Back in the *File system* dialog:
    - Select only the **ex07** and **ut07** files as listed from the folder:
      - ex07\_createCluster.py
      - ex07\_createClusterMember.py
      - ex07\_queryClusterState.py
      - ex07\_startCluster.py
      - ex07\_stopCluster.py
      - ut07\_createCluster.py
      - ut07\_createClusterMember.py
      - ut07\_queryClusterState.py
      - ut07\_startCluster.py
      - ut07\_stopCluster.py

- Type **/Scripts** at the end of **PlantsScriptingProject** in the Into folder field. This action results in the files that are imported in a folder named *Scripts* under the project.



- Click **Finish**.

- \_\_\_ g. In the Enterprise Explorer view, expand **PlantsScriptingProject > Scripts**. The skeleton function and mainline scripts for this exercise is displayed in the folder.
- \_\_\_ 5. Import the property file that contains the Plants configuration properties.
- \_\_\_ a. In the Enterprise Explorer view, right-click **PlantsScriptingProject** and select **Import**.

- \_\_\_ b. In the *Import select* dialog, expand **General** and select **File system**.
- \_\_\_ c. Click **Next**.
- \_\_\_ d. In the *File system* dialog, click **Browse** next to the *From directory* field.
- \_\_\_ e. In the *Import from directory* dialog, go to **/usr/Software/Scripts**, select **Properties**, and click **OK**.
- \_\_\_ f. Back in the *File system* dialog:
  - Select the **Properties** check box to select the single property file that the folder contains, **plants.properties**.
  - Type **/Properties** at the end of PlantsScriptingProject in the *Into folder* field. This actions results in the files that are imported in a folder named *Properties* under the project.



- Click **Finish**.
- \_\_\_ g. In the Enterprise Explorer view, expand **PlantsScriptingProject > Properties**. The **plants.properties** file is displayed in the folder.

### Section 3: Read-only: Create a server definition for the deployment manager and start the server.

 Warning

#### Federated servers are not supported in the IBM Assembly and Deploy Tools

Managed or federated servers, including the deployment manager, are not supported in Rational Application Developer or the IBM Assembly and Deploy Tools for WebSphere Administration (IADT).

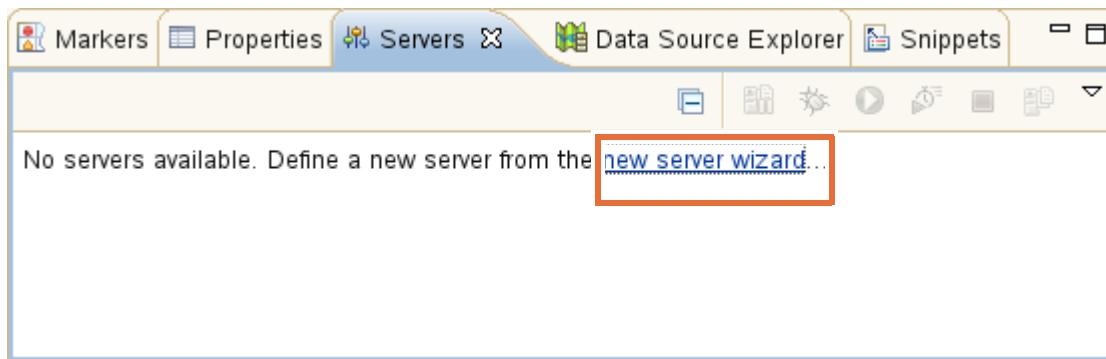
If you choose to complete the steps in this section, the following problems occur when using the server within the IBM Assembly and Deploy Tools.

- You cannot start the server (dmgr) from within IADT. It must be started from the command line.
- You cannot make any configuration changes to the server definition from within IADT.
- There is no Console view available for the server within IADT.
- Performance degrades because the IADT tries continuously to publish the UTC application, and fails.

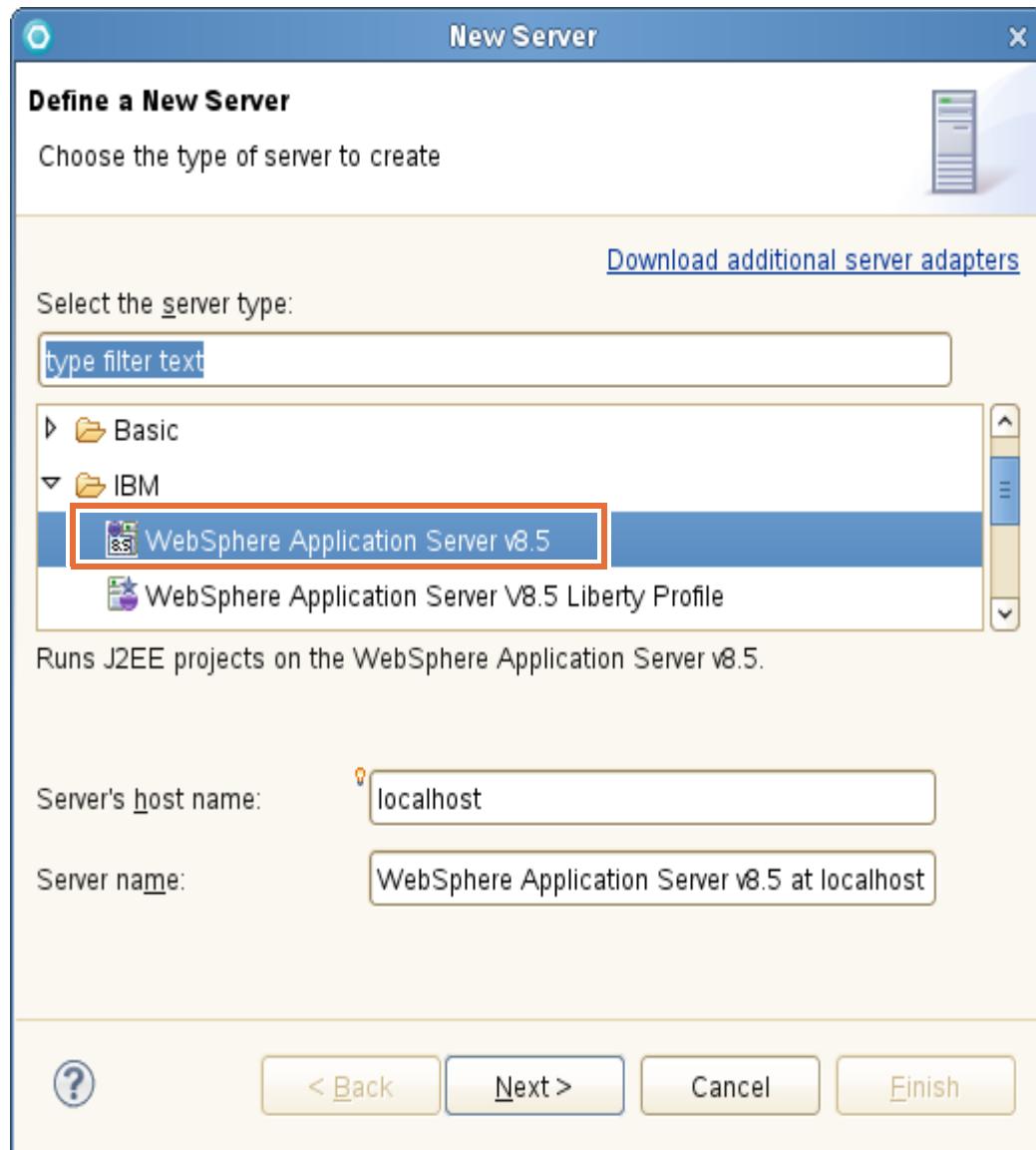
Though this section is provided for reference, you are encouraged to skip it and proceed to the next section, **Section 4: Understanding your development environment**.

For the next several exercises, scripts are run against the deployment manager from the command line rather than by using the `scriptExecutor.py` utility as you did in the earlier exercises.

- \_\_\_ 1. Create a server definition for the dmgr server.
  - \_\_\_ a. Click the **Servers** tab to gain focus on the Servers view.
  - \_\_\_ b. Click the link **new server wizard**.

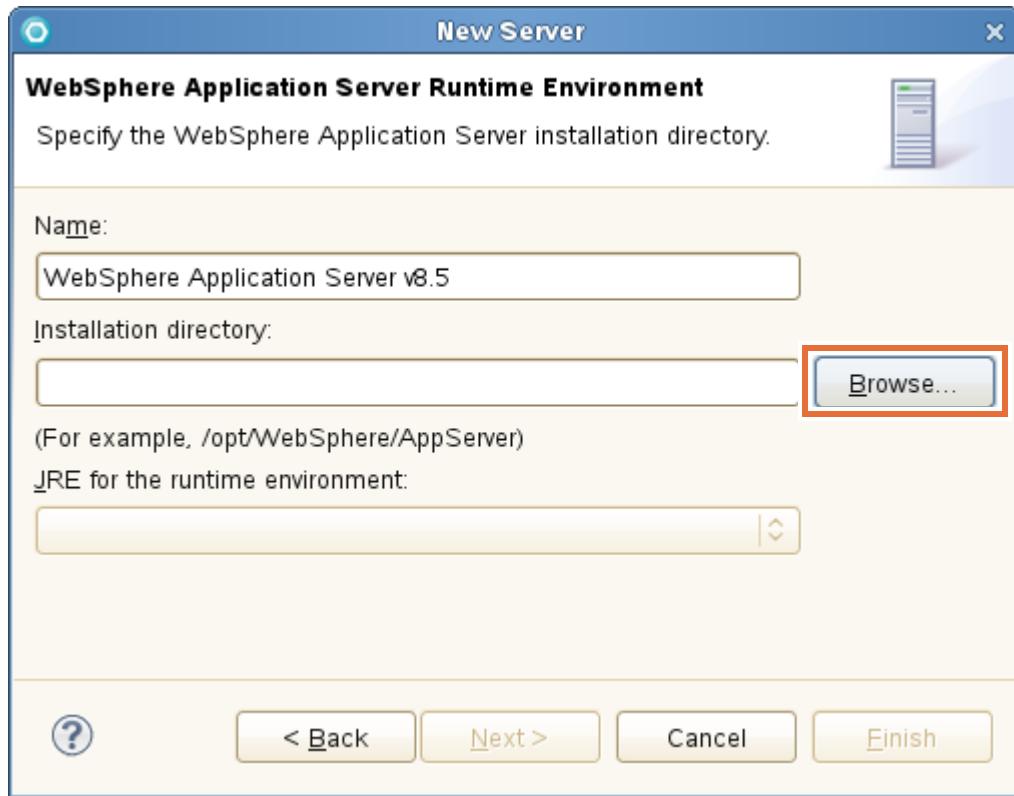


- \_\_\_ c. In the *Define a New server* dialog, scroll down, expand IBM, and select **WebSphere Application Server v8.5**



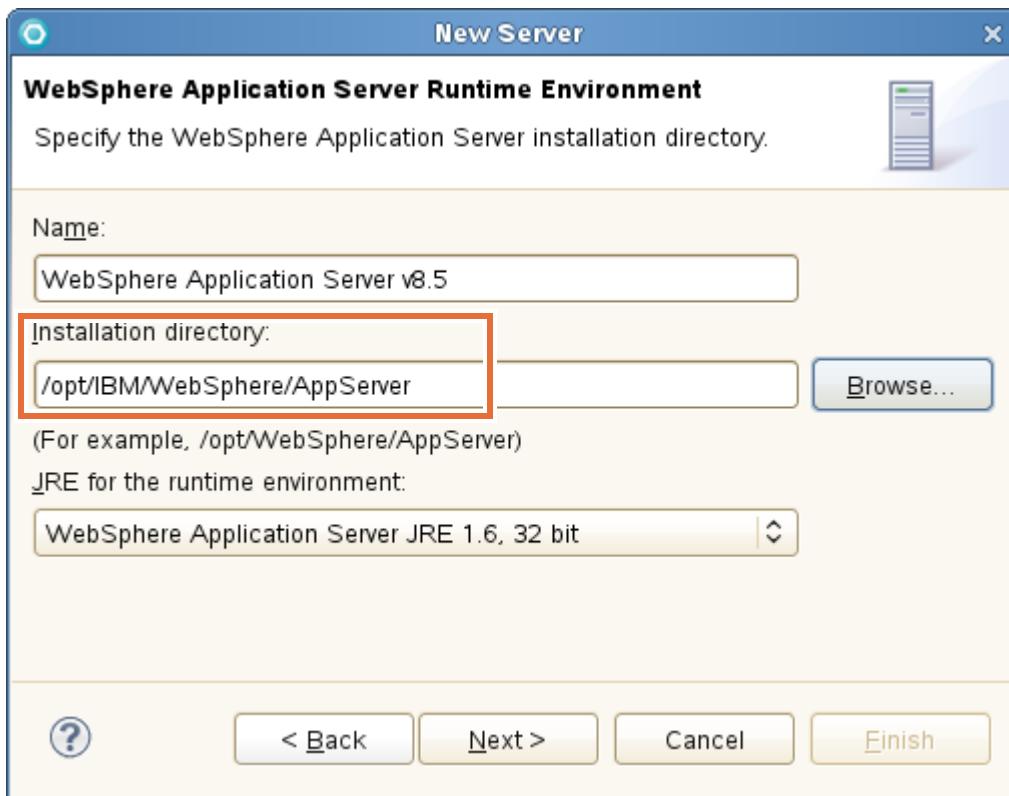
- \_\_\_ d. Click **Next**.

- \_\_\_ e. In the *WebSphere Application Server Runtime Environment* dialog, click **Browse** next to the Installation directory field.



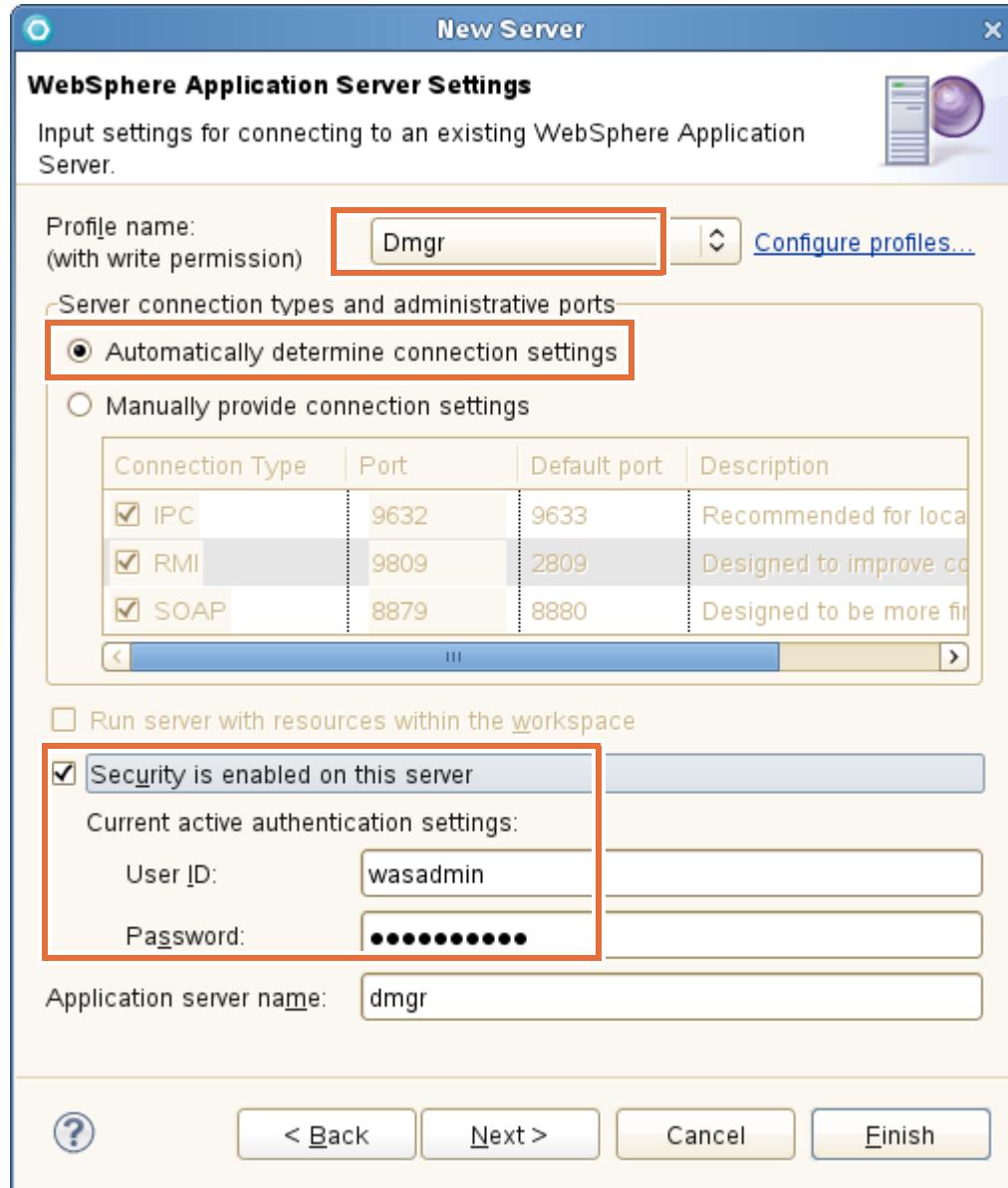
- \_\_\_ f. Go to **/opt/IBM/WebSphere**, and select **AppServer** and click **OK**.

- g. In the *WebSphere Application Server Runtime Environment* dialog, the Installation directory field is now properly specified. Click **Next** to continue.

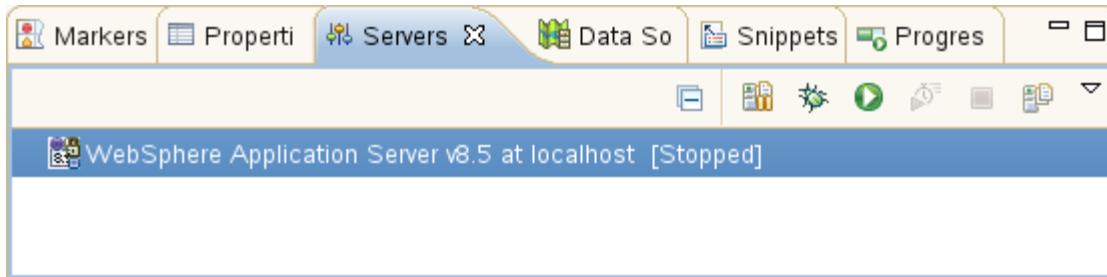


- h. In the *WebSphere Application Server Settings* dialog:
- Since administrative security is enabled on the server, enter `wasadmin` in the **User ID** field and `web1sphere` in the **Password** field.

- Keep the default for Server connection types..



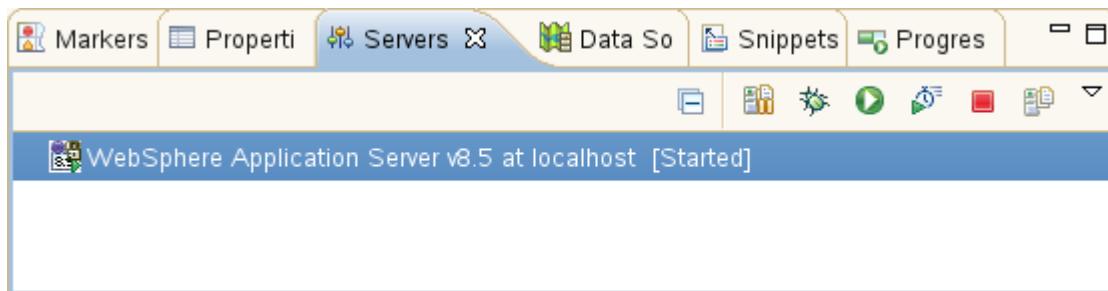
- Click **Finish**. After a moment, the deployment manager server is displayed in the Servers view as **WebSphere Application Server v8.5 at localhost** and shows a status of **Stopped**.



2. Start the dmgr server from the command line.
- \_\_ a. In a terminal window, go to <profile\_root>/Dmgr/bin
  - \_\_ b. Enter the command: ./startmanager.sh
  - \_\_ c. Wait until the server is started.
  - \_\_ d. Back in the Server view of the IADT, click **Start Server**.
  - \_\_ e. If you see the following error message, click **OK**.



- \_\_ f. Verify that the server status is now **Started**.

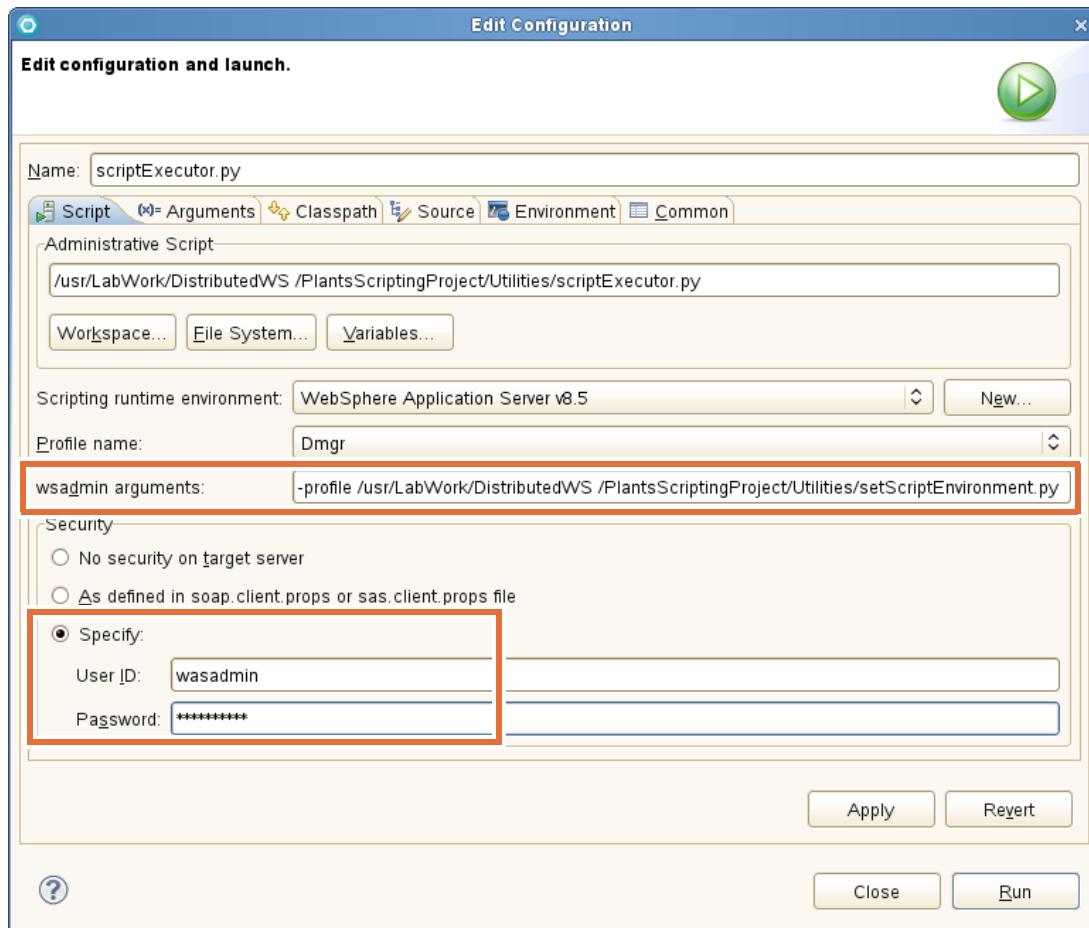


3. Run the scriptExecutor utility script.
- \_\_ a. In the Enterprise Explorer view, expand **PlantsScriptingProject > Utilities** if necessary. Right-click **scriptExecutor.py** and select **Run As > Administrative Script**.
  - \_\_ b. Since you are running the script for the first time, the **Edit configuration and launch** window opens so that you can specify the run configuration parameters. In the dialog:
    - Select **WebSphere Application Server v8.5** in the drop-down list for Scripting runtime environment.
    - Select **Dmgr** in the drop-down list for Profile name.
    - In the wsadmin arguments field, enter (as a single line):

```
-profile
/usr/LabWork/DistributedWS/PlantsScriptingProject/Utilities/setScriptEnvironment.py
```

This configuration causes wsadmin to run the indicated profile script on startup. The purpose of the *setScriptEnvironment.py* profile script is explained in detail later in the step.

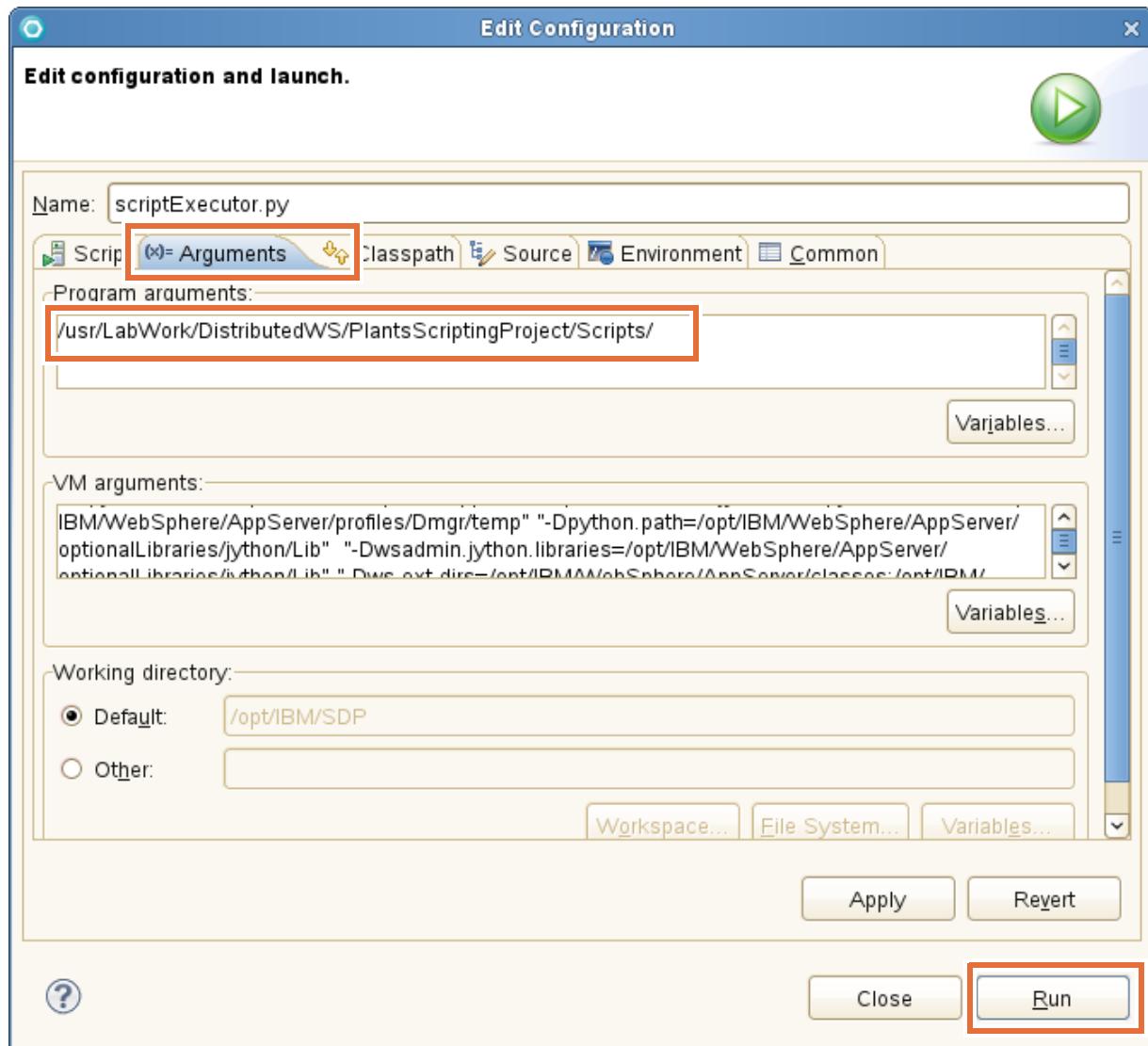
- In the Security section, select the **Specify** radio button and type `wasadmin` for the **User ID** field and `web1sphere` for the **Password** field.



- Select the **(x)=Arguments** tab to specify an argument to pass to the scriptExecutor script.
- \_\_\_ c. On the **(x)=Arguments** tab, type `/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/` in the Program arguments field. This argument specifies the location where the scriptExecutor looks for scripts to run.

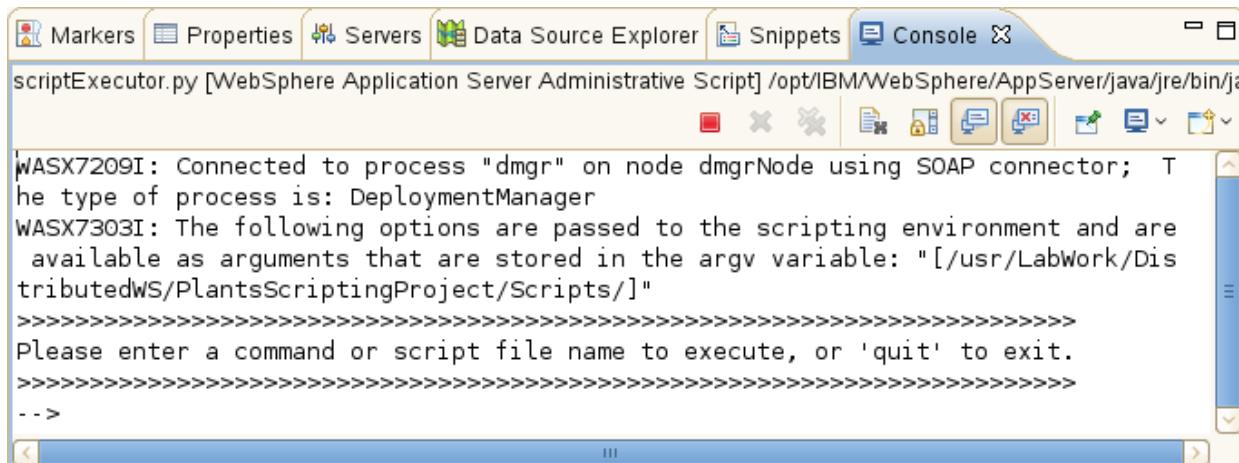
**Important**

Make sure to type forward slashes and include the last slash.



- \_\_\_ d. Click **Run** to run the script. After a few moments, the **Console** view displays a message that confirms the establishment of a wsadmin connection with the *Dmgr* server. Since you are running it against the *Dmgr* profile for the first time, wsadmin displays a series of messages that indicate the loading of required Jython libraries (\*sys-package-mgr\*: processing new jar... messages).

Finally, an input prompt is displayed for running an administrative command or a script file.



## Note

You have two console views opened: one for the *scriptExecutor* script and the other of the *dmgr* standard output. To switch between console views, click the drop-down arrow of the **Display Selected Console** button and select the console output to view.

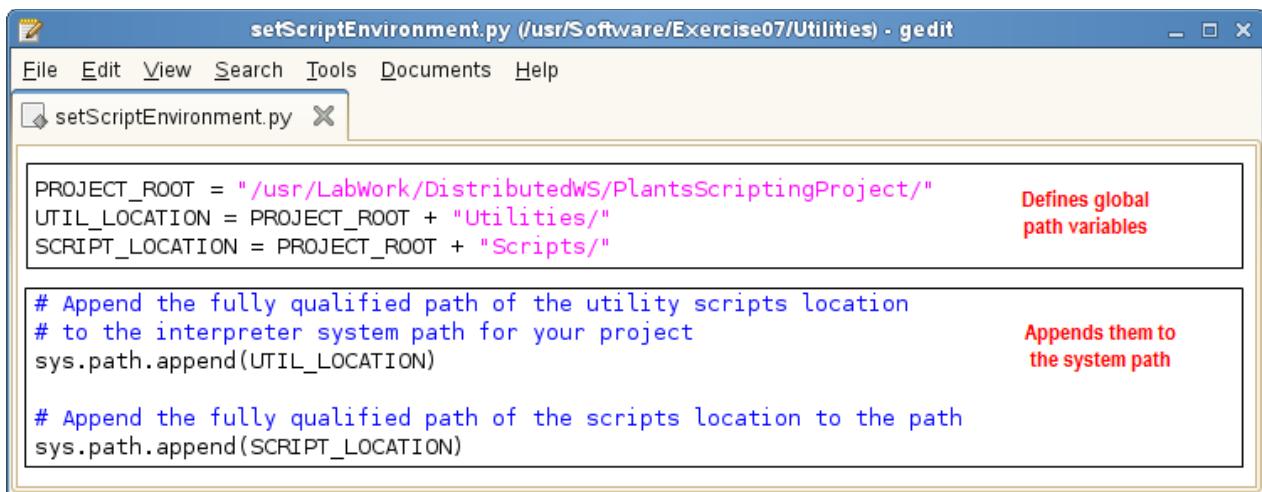
- \_\_\_ 4. Your script development environment is now set up. However, to effectively use it, you must better understand the artifacts that it contains and how to use them. You also must know what is expected in terms of how you develop your scripts. This information is provided in the “Important” box. Take a moment to read and understand its contents.

## Section 4: Understanding your development environment

The following paragraphs explain the content and intended use of the utility scripts and helper files that are used in this exercise.

### setScriptEnvironment.py

This utility script creates three global environment variables, PROJECT\_ROOT, UTIL\_LOCATION and SCRIPT\_LOCATION, representing the path to the location of the Jython project, to the **Utilities** folder, and to the **Scripts** folder in the workspace. It then appends the UTIL\_LOCATION and SCRIPT\_LOCATION paths to the interpreter system path so that references to developed scripts and provided utilities can be resolved without qualifying their full path name. This script is intended to be run as a *profile script* argument on the command-line invocation of wsadmin. Therefore, the variables have global scope within the session, and the system path is set up once at the beginning of the session.



```

setScriptEnvironment.py (/usr/Software/Exercise07/Utilities) - gedit
File Edit View Search Tools Documents Help
setScriptEnvironment.py X

PROJECT_ROOT = "/usr/LabWork/DistributedWS/PlantsScriptingProject/"
UTIL_LOCATION = PROJECT_ROOT + "Utilities/"
SCRIPT_LOCATION = PROJECT_ROOT + "Scripts/"

# Append the fully qualified path of the utility scripts location
# to the interpreter system path for your project
sys.path.append(UTIL_LOCATION)

# Append the fully qualified path of the scripts location to the path
sys.path.append(SCRIPT_LOCATION)

```

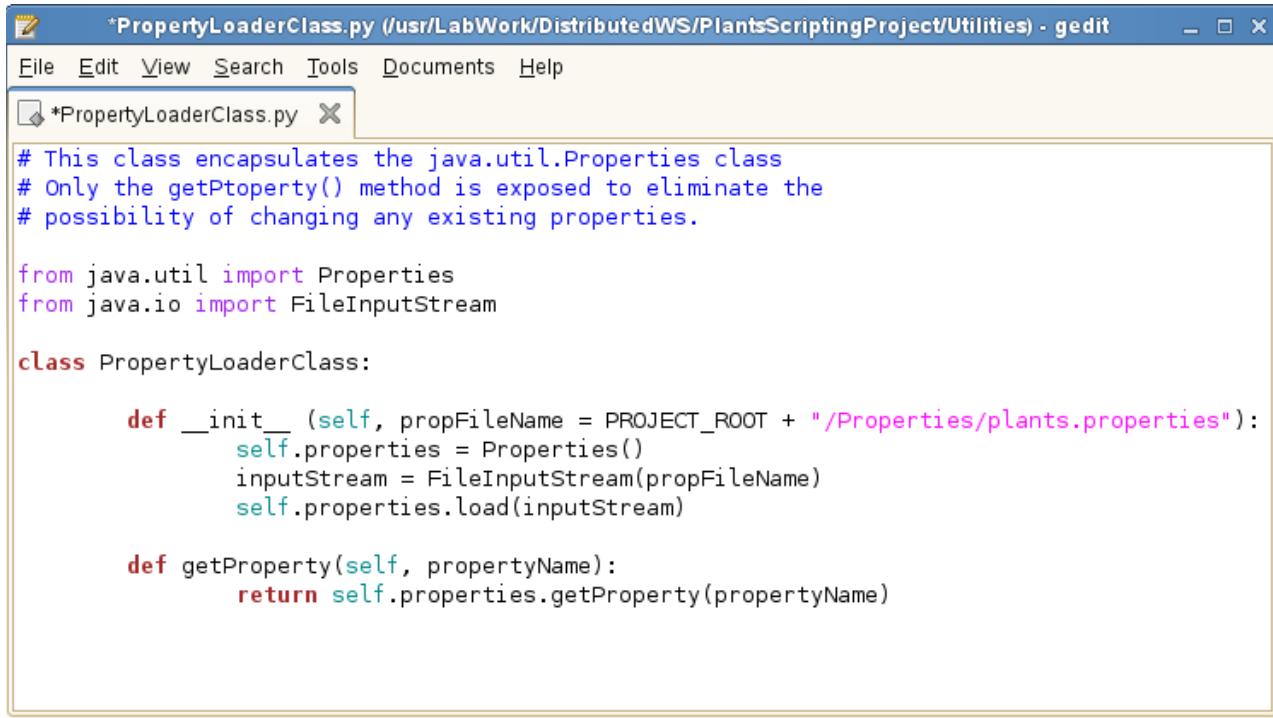
**Defines global path variables**

**Appends them to the system path**

### PropertyLoaderClass.py

This utility Jython class enables a script to retrieve property values from an external property file. It uses the `java.util.Properties` class to load the external file whose name is passed as a constructor argument, and encapsulates it so that property values

can be read only and not modified. Scripts that need read access to a property file can create an instance of this class and use its `getProperty(aPropertyName)` method.



```
*PropertyLoaderClass.py (/usr/LabWork/DistributedWS/PlantsScriptingProject/Utilities) - gedit
File Edit View Search Tools Documents Help
*PropertyLoaderClass.py X

# This class encapsulates the java.util.Properties class
# Only the getProperty() method is exposed to eliminate the
# possibility of changing any existing properties.

from java.util import Properties
from java.io import FileInputStream

class PropertyLoaderClass:

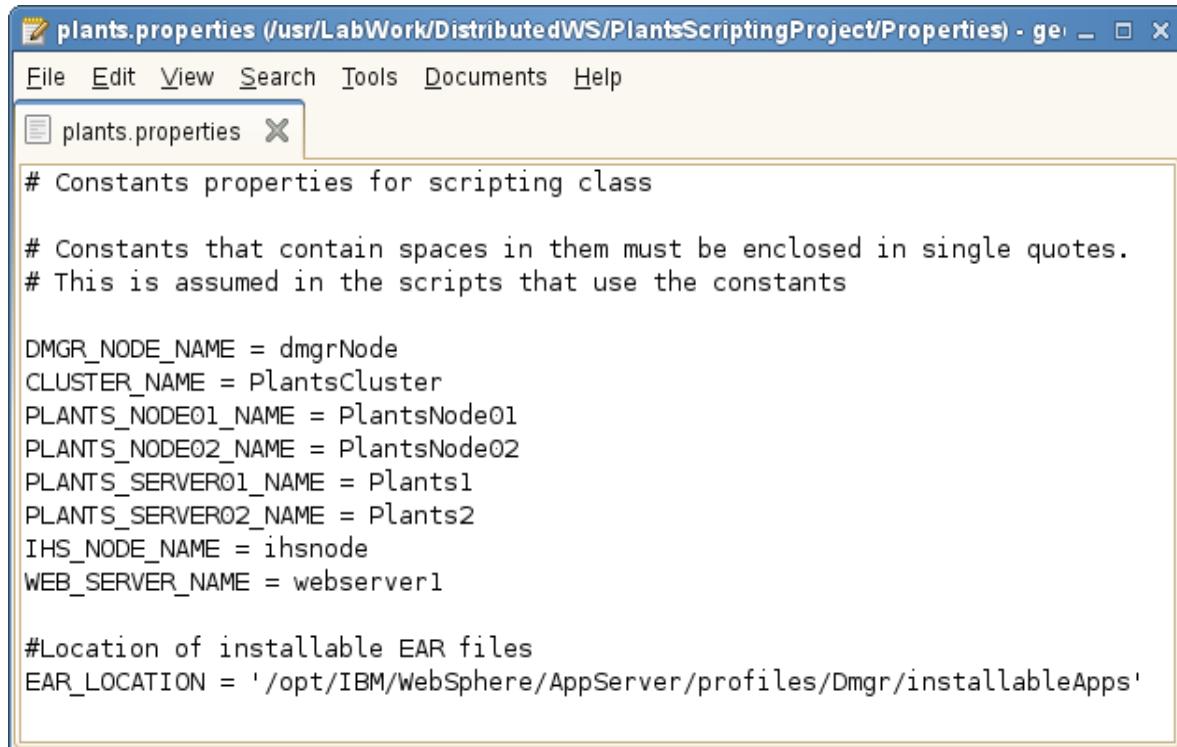
    def __init__(self, propFileName = PROJECT_ROOT + "/Properties/plants.properties"):
        self.properties = Properties()
        inputStream = FileInputStream(propFileName)
        self.properties.load(inputStream)

    def getProperty(self, propertyName):
        return self.properties.getProperty(propertyName)
```

## plants.properties

This text file contains the constant values for the configuration properties of the PlantsByWebSphere applications. Each property is defined on a separate line by using the format: *name = value*. As a good programming (and scripting) practice, you should avoid hardcoding literals inside your code. Instead, define them in a property file, and

retrieve them from the file when needed in the code. The properties file promotes code reuse and flexibility, and simplifies code maintenance.



```

plants.properties (/usr/LabWork/DistributedWS/PlantsScriptingProject/Properties) - ge... X
File Edit View Search Tools Documents Help
plants.properties X

# Constants properties for scripting class

# Constants that contain spaces in them must be enclosed in single quotes.
# This is assumed in the scripts that use the constants

DMGR_NODE_NAME = dmgrNode
CLUSTER_NAME = PlantsCluster
PLANTS_NODE01_NAME = PlantsNode01
PLANTS_NODE02_NAME = PlantsNode02
PLANTS_SERVER01_NAME = Plants1
PLANTS_SERVER02_NAME = Plants2
IHS_NODE_NAME = ihsnode
WEB_SERVER_NAME = webserver1

#Location of installable EAR files
EAR_LOCATION = '/opt/IBM/WebSphere/AppServer/profiles/Dmgr/installableApps'

```

## **exNN\_<scriptName>.py and utNN\_<scriptName>.py**

These script files contain initial skeleton code for developing the wanted capability. They supply instruction comments and code snippets to help guide you in writing your code. They are also provided to illustrate the good script design practice of separating the script that implements the wanted capability from the one that is used to unit test it or start it.

For each administrative function that you implement, you develop two script files:

**exNN\_<scriptName>.py:** This script implements the wanted functions as a Jython function that accepts its input values as parameters. The initial skeleton file provides a function definition statement and general instructions in the form of comments to help complete the code. It also includes basic print statements

that display execution progress messages. An example function skeleton script file is shown.

```

*ex07_createCluster.py (/usr/Software/Exercise07/Scripts) - gedit
File Edit View Search Tools Documents Help
*ex07_createCluster.py X

# ex07_createCluster.py

def createCluster(clusterName,
                  preferLocal="true",
                  createReplicationDomain="true"):

    print "Creating cluster"

    #Build options list for -clusterConfig step.
    clusterConfigOptions = []

    #Build options list for -replicationDomain step.
    replicationDomainOptions = []

    # Call AdminTask to create the cluster.
    AdminTask.

    # Save the configuration change.
    #
    #AdminConfig.save()

    print "Cluster created"

```

**Function definition with pre-filled parameter**

**Instruction comments indicating code that must be completed**

**Code snippets to help script development**

**AdminConfig.save() initially commented out for testing**



### Note

The function skeleton script contains an *AdminConfig.save()* method invocation line that is initially commented out. You should not save any change to the configuration repository while you are developing and testing the script. After you are done testing and are satisfied that it works correctly, you must uncomment this line.

***utNN\_<scriptName>.py:*** This script starts the function that is contained in the other file in order to run it. It first retrieves any required configuration values from the *plants.properties* and supplies them as parameters in the function invocation line. The initial skeleton file already includes code to create an instance of the *PropertyLoaderClass*, assigned to a variable named *gp* (for *global properties*),

and code snippets to get property values and start the wanted function. An example unit test script file is shown.

```

*ut07_createCluster.py (/usr/Software/Exercise07/Scripts) - gedit
File Edit View Search Tools Documents Help
*ut07_createCluster.py X
# ut07_createCluster.py

# Load the PropertyLoaderClass and the script function to be called.
#
execfile(UTIL_LOCATION + "/PropertyLoaderClass.py") ← Loads required class and
execfile(SCRIPT_LOCATION + "/ex07_createCluster.py")   ← function modules

"""
You may instantiate the Property Loader class with your own properties file
by calling it like this: gp=PropertyLoaderClass(propertyFile)
where "property file" is a fully qualified path to your properties file.
Not passing the name of the property file, as shown below, applies the default
property file name as coded in the class definition (see PropertyLoaderClass.py)
"""

gp=PropertyLoaderClass() ← Creates an instance of
# Write code to test function below.

# Get the required properties.
#
# Get the cluster name property.
clusterName = gp.getProperty("") ← Provides a template for
# retrieving property values

#Call the createCluster()function.
#
createCluster() ← Provides a template for invoking
                  the function to be tested

```

This design promotes script reusability, as the parameterized script that implements the administrative function can be used and started for different applications. In fact, the intent is to allow you to reuse the scripts that you develop in this exercise in your own production environment without any change.

Figure 2 illustrates how the utility scripts and helper files work together to give you a flexible and reusable script design architecture.

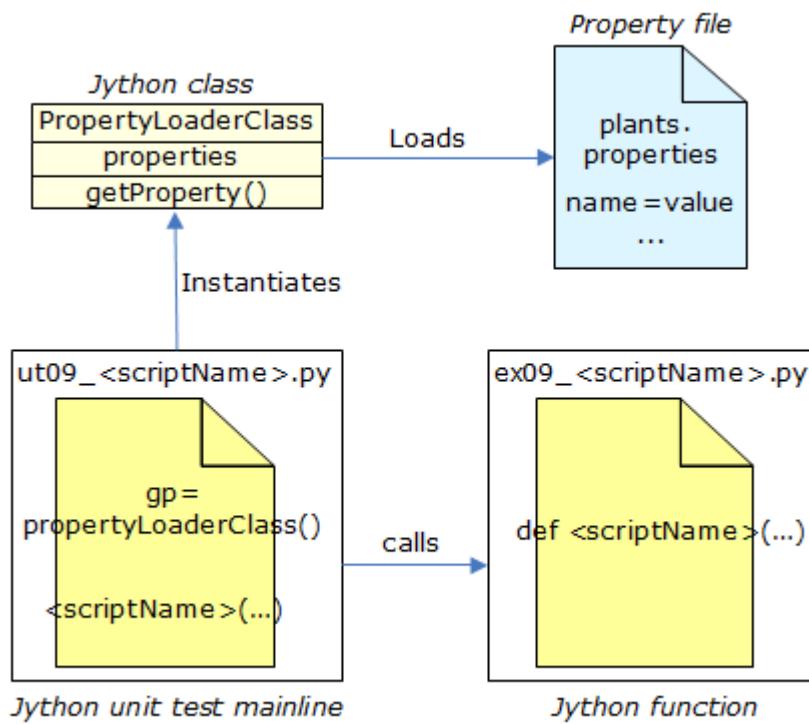


Figure 2 - Plants script design architecture

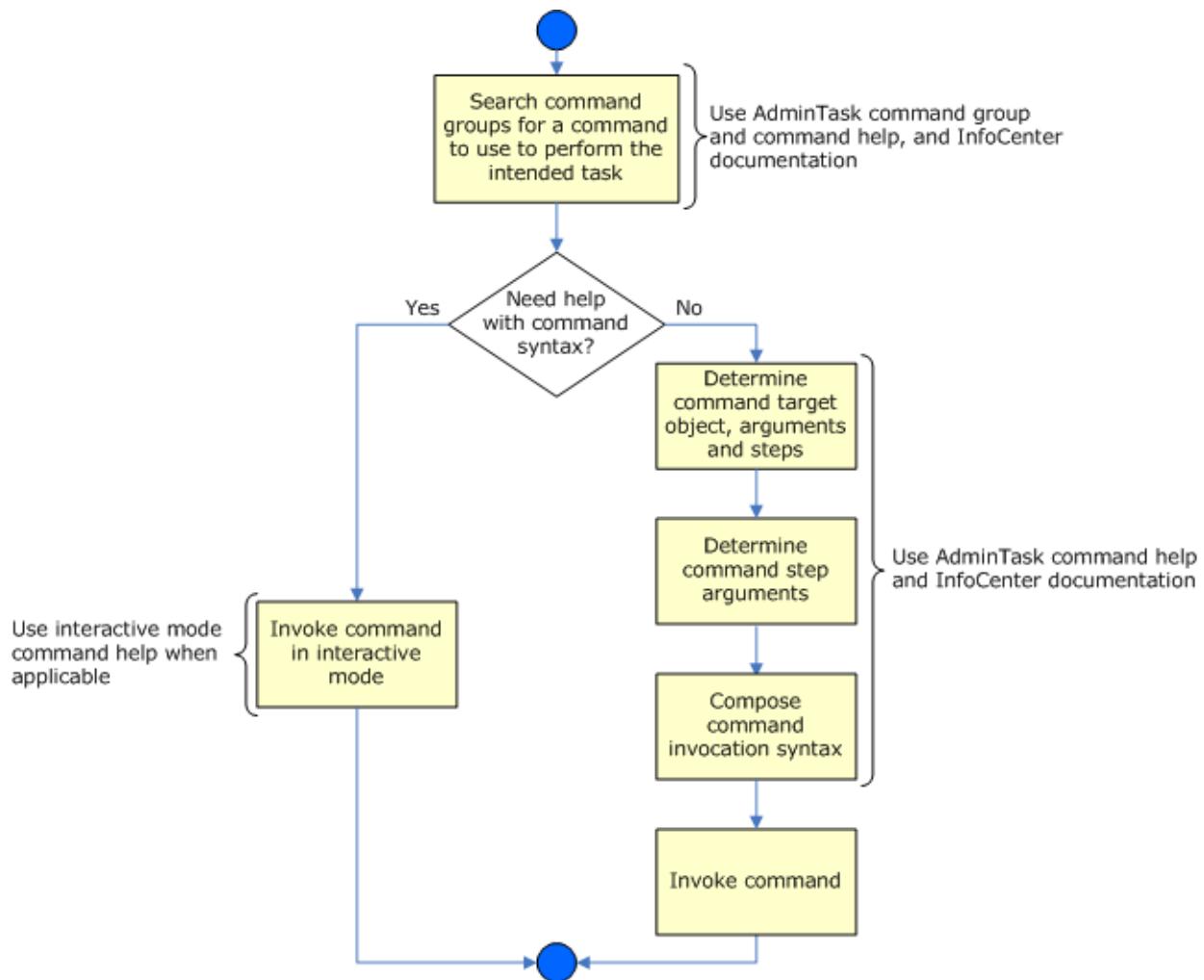
## How to develop your scripts

In this exercise, little instruction is given in terms of how to use the administrative objects because you covered that in the individual administrative object labs. In fact, the general content of the instructions consists of:

1. Requirements specification for the script to develop
2. References to the function and unit test script files to use as a starting point for development
3. Hints as applicable

You are expected to follow the general instructions that are found in the comments of the skeleton files that provide an outline for writing your code. You must also research the required administrative object methods and syntax on your own, using the knowledge that you gained from the previous lectures and labs. Therefore, as you write your script, refer to these approaches, techniques, and resources as needed. For example, to find out how to

use the AdminTask object to perform a wanted function, refer to the following figure, found in Exercise 6, which provides help on required steps and available resources:



### Important

## Solution scripts

Finally, if you get stuck or want only to run the completed version of a script, you can import the solution script from the `/usr/Solutions/Exercise07` folder and run it in the IADT.

## Section 5: Creating the Plants cluster

In this section, you first develop a parameterized Jython function that creates a cluster by using the AdminTask object. You then start it to create the *PlantsCluster* as depicted in Figure 1.

### Requirements specification

Your specific requirements are as follows:

1. Create a Jython function named `createCluster` to create a cluster given the following configuration options as parameters:
  - `clusterName`: Name of cluster to create
  - `preferLocal`: Boolean flag to indicate whether to enable or disable node scoped routing optimization within this cluster.
  - `createReplicationDomain`: Boolean flag to indicate whether to create a replication domain for HTTP session data replication for the cluster.
2. Start the function to create the PlantsCluster, with the following options:

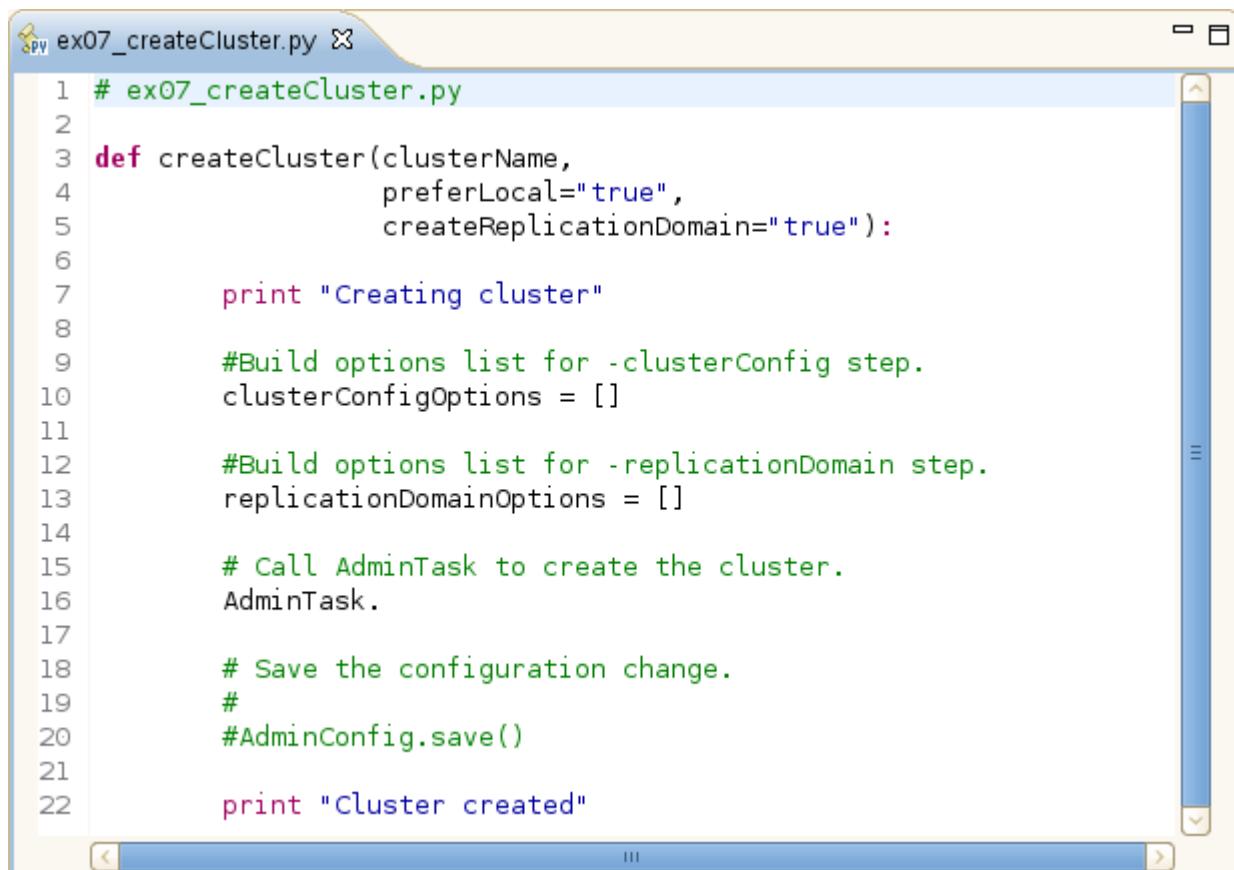
**Table 3: Options for PlantsCluster**

	<b>Option name</b>	<b>Option value</b>
1	<code>clusterName</code>	Retrieve the value of property named "CLUSTER NAME" from plants.properties.
2	<code>preferLocal</code>	"true"
3	<code>createReplicationDomain</code>	"true"

### Creating the `createCluster` function

1. A skeleton script, `ex07_createCluster.py`, is provided to get you started. It is located under **PlantsScriptingProject > Scripts** in the Enterprise Explorer view. Open it in the Jython editor.

This file provides a function definition statement and general instructions in the form of comments to help complete the code. It also includes basic print statements that display execution progress messages.



```

1 # ex07_createCluster.py
2
3 def createCluster(clusterName,
4                   preferLocal="true",
5                   createReplicationDomain="true"):
6
7     print "Creating cluster"
8
9     #Build options list for -clusterConfig step.
10    clusterConfigOptions = []
11
12    #Build options list for -replicationDomain step.
13    replicationDomainOptions = []
14
15    # Call AdminTask to create the cluster.
16    AdminTask.
17
18    # Save the configuration change.
19    #
20    #AdminConfig.save()
21
22    print "Cluster created"

```

- 2. To complete the instructions in lines 9, 12 and 15, you must identify the appropriate AdminTask command to use, understand its target, arguments, and steps, and compose its invocation syntax. You learned how to find this information in *Exercise 6- Using the AdminTask object* and *Unit 11 - AdminTask basics*, and can refer to these resources if necessary. In particular, the flowchart on *How to use AdminTask object* found in both these resources can be of good help. Using what you learned, complete the code as instructed by lines 9, 12 and 15.

**Hint**

The **ClusterConfigCommands** command group is a good place to start your search for an AdminTask command to create a cluster member. Use it as a keyword to search the Information Center and drill down to a possible match.

[Network Deployment \(Distributed operating systems\), Version 8.5](#) > [Reference](#) > [Commands \(wsadmin scripting\)](#)

## **ClusterConfigCommands command group for the AdminTask object**

You can use the Jython or Jacl scripting languages to cluster application servers, generic servers, and proxy servers using scripting. The commands and parameters in the **ClusterConfigCommands** group can be used to create and delete server clusters and servers known as cluster members.

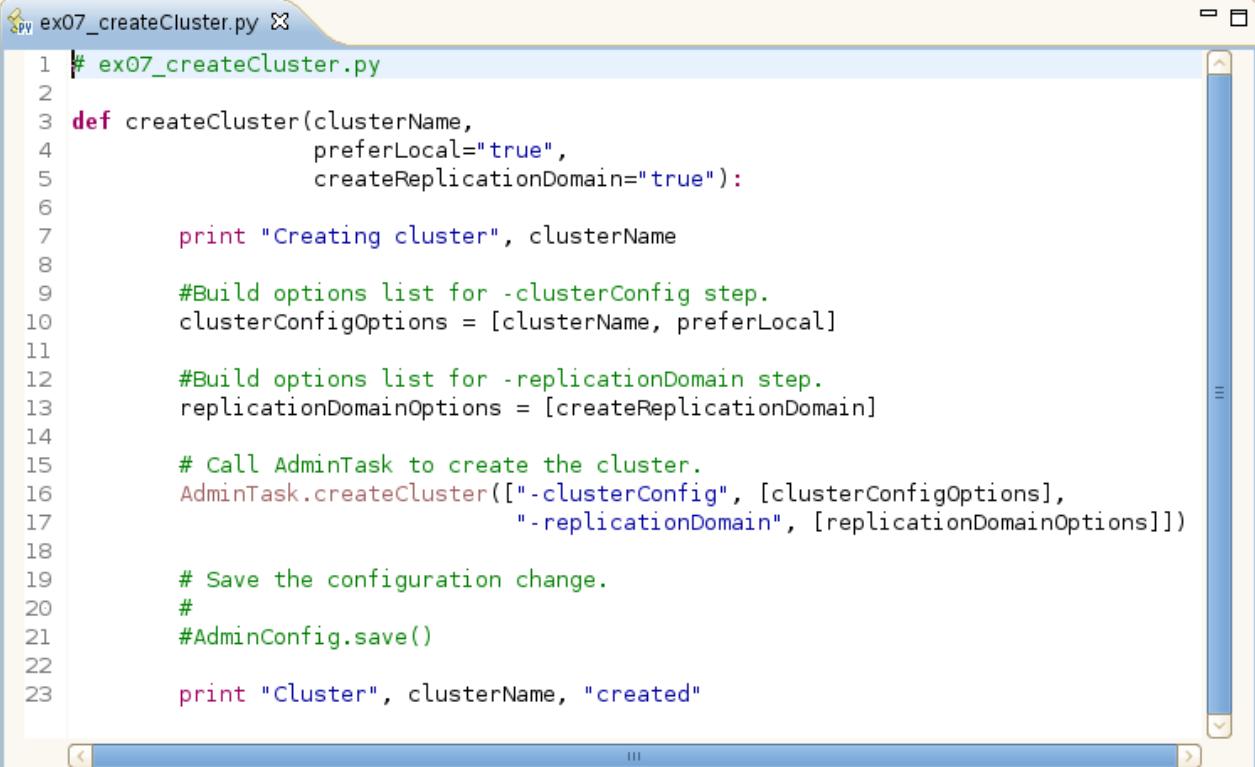
The **ClusterConfigCommands** command group for the AdminTask object includes the following commands:

- [createCluster](#)
- [createClusterMember](#)
- [deleteCluster](#)
- [deleteClusterMember](#)

After you identify the command to use, use the description and examples that are provided in the Information Center to figure out the target, arguments, and steps that it requires. You can also get more usage details by using one or more of the following methods:

- *AdminTask.help(aCommandName)*
- *AdminTask.help(aCommandName, aStepName)*
- *AdminTask.aCommandName("-interactive")* (runs the command in interactive mode)

Your completed *createCluster* function should look as follows:



```

1 # ex07_createCluster.py
2
3 def createCluster(clusterName,
4                   preferLocal="true",
5                   createReplicationDomain="true"):
6
7     print "Creating cluster", clusterName
8
9     #Build options list for -clusterConfig step.
10    clusterConfigOptions = [clusterName, preferLocal]
11
12    #Build options list for -replicationDomain step.
13    replicationDomainOptions = [createReplicationDomain]
14
15    # Call AdminTask to create the cluster.
16    AdminTask.createCluster(["-clusterConfig", [clusterConfigOptions],
17                           "-replicationDomain", [replicationDomainOptions]])
18
19    # Save the configuration change.
20    #
21    #AdminConfig.save()
22
23    print "Cluster", clusterName, "created"

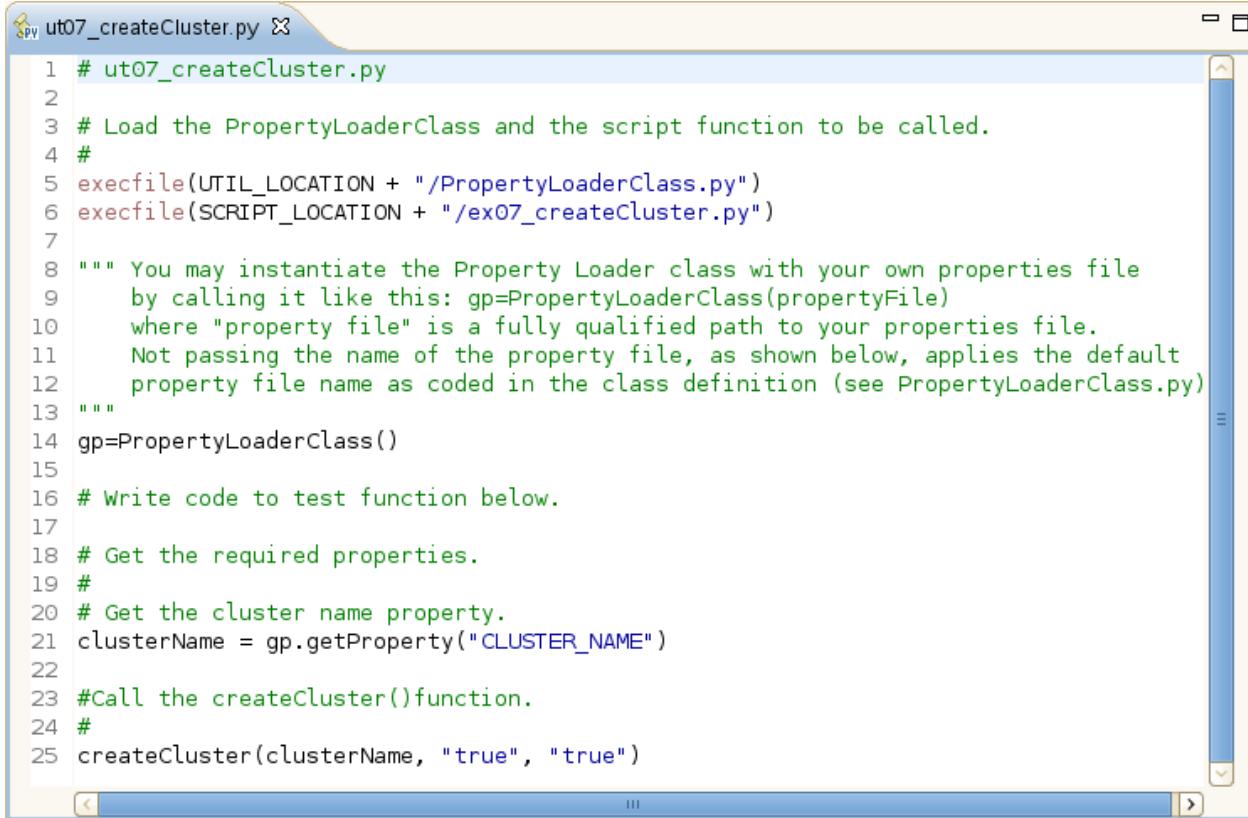
```

- \_\_\_ 3. Save your changes.

## Starting the *createCluster* function

- \_\_\_ 1. Use the **ut07\_createCluster.py** script to start the *createCluster* function that you created. Modify it to:
  - \_\_\_ a. Retrieve any properties that are required to create the *PlantsCluster* as listed in the options table, by using the `gp.getProperty(propertyName)` method. In this case, only the "CLUSTER\_NAME" property is needed.
  - \_\_\_ b. Start the *createCluster* function to create *PlantsCluster* with the parameter values specified in the options table.

Your **ut07\_createCluster.py** script should look as follows:



```

1 # ut07_createCluster.py
2
3 # Load the PropertyLoaderClass and the script function to be called.
4 #
5 execfile(UTIL_LOCATION + "/PropertyLoaderClass.py")
6 execfile(SCRIPT_LOCATION + "/ex07_createCluster.py")
7
8 """ You may instantiate the Property Loader class with your own properties file
9 by calling it like this: gp=PropertyLoaderClass(propertyFile)
10 where "property file" is a fully qualified path to your properties file.
11 Not passing the name of the property file, as shown below, applies the default
12 property file name as coded in the class definition (see PropertyLoaderClass.py)
13 """
14 gp=PropertyLoaderClass()
15
16 # Write code to test function below.
17
18 # Get the required properties.
19 #
20 # Get the cluster name property.
21 clusterName = gp.getProperty("CLUSTER_NAME")
22
23 #Call the createCluster()function.
24 #
25 createCluster(clusterName, "true", "true")

```

- \_\_\_ 2. Save **ut07\_createCluster.py**.
- \_\_\_ 3. If the deployment manager is not already started, start it now.
  - \_\_\_ a. Using the terminal window, go to the <profile\_root>/Dmgr/bin directory.
  - \_\_\_ b. Enter the command: **./startManager.sh**
  - \_\_\_ c. Wait for the deployment manager to start and record its process ID \_\_\_\_\_.
- \_\_\_ 4. Run **ut07\_createCluster.py** from a terminal window.
  - \_\_\_ a. Go to **<profile\_root>/Dmgr/bin**.
  - \_\_\_ b. Enter the following command all on one line or copy it from the **/usr/Solutions/Exercise07/cmd.txt** file:
 

```
./wsadmin.sh -user wasadmin -password web1sphere -profile /usr/LabWork/DistributedWS/PlantsScriptingProject/Utilities /setScriptEnvironment.py -f /usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut07_createCluster.py
```
  - \_\_\_ c. Examine the output from this command in the terminal window. Correct any errors that might show up and rerun until successful.

- \_\_\_ d. When you are satisfied that your script works, edit `ex07_createCluster.py` to uncomment the `AdminConfig.save()` line; then save and unit test it again by running `ut07_createCluster.py` from a terminal window.



### Important

Make sure to run the script again with the `AdminConfig.save()` line uncommented; otherwise, you do not see the effect of your changes when you verify the results in the next section.

## Verifying the results

- \_\_\_ 1. Verify that the *PlantsCluster* was created.
  - \_\_\_ a. From a terminal window, go to `<profile_root>/Dmgr/bin`.
  - \_\_\_ b. Enter the command: `./wsadmin.sh -conntype none -lang jython`
  - \_\_\_ c. At the wsadmin prompt, enter: `print AdminConfig.list("ServerCluster")`

```

Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./wsadmin.sh -conntype none -lang jython
WASX7357I: By request, this scripting client is not connected to any server process. Certain configuration and application operations will be available in local mode.
WASX7031I: For help, enter: "print Help.help()"
wsadmin>print AdminConfig.list("ServerCluster")
PlantsCluster(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#ServerCluster_1360704288099)
wsadmin>quit
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin #
  
```

The command should return the configuration name of the *PlantsCluster*.

- \_\_\_ d. Type **quit** to exit the wsadmin session.
- \_\_\_ 2. Close all open Jython editor views in IADT.

You successfully developed a Jython script to create a cluster.



### Information

The following section is optional and can be skipped if you want.

## Handling error situations (optional)

Enhance your script so that it gracefully handles the situation when the user is attempting to create a cluster that exists in the configuration. In such a case, delete the existing cluster first, then create it again. Are there any other dependencies that you should address also? (Hint: what about the data replication domain that can be created along with the cluster?) The following code listing shows an example of how you might implement this enhancement:

```
def createCluster(clusterName,
                 preferLocal="true",
                 createReplicationDomain="true"):

    Code to handle existing cluster and data replication domain condition
    # Check if cluster already exists. If so, delete it first.
    if AdminConfig.getId("/ServerCluster:" + clusterName + "/") != "":
        print "Cluster", clusterName, "already exists. Deleting existing one"

        # Check to see if corresponding data replication domain also exists and
        # needs to be deleted.
        if AdminConfig.getId("/DataReplicationDomain:" + clusterName + "/") != "":
            print "Deleting corresponding data replication domain also"
            deleteDRD = "true"
        else:
            deleteDRD = "false"

        AdminTask.deleteCluster(["-clusterName", clusterName,
                               "-replicationDomain", [[deleteDRD]]])

        print "Existing", clusterName, "deleted"

    print "Creating cluster"
```

## Section 6: Creating the Plants cluster members

Now that you created the *PlantsCluster*, you are ready to create its cluster members. In this step, you first develop a parameterized Jython function that creates a cluster member by using the AdminTask object. You then start it twice to create the *Plants1* and *Plants2* application servers as members of the *PlantsCluster*, as depicted in Figure 1.



### Information

Servers have to be created as cluster members from the start; they cannot be added to a cluster after they are created.

Either the AdminTask or AdminConfig object can be used to create a cluster member. In general, when given the choice, using the AdminTask object is usually easier and that is what is used in this step.

## Requirements specification

Your specific requirements are as follows:

1. Create a Jython function named `createClusterMember` to create a cluster member given the following configuration options as parameters:
  - `clusterName`: Name of cluster member to create
  - `memberNode`: Name of node that the new cluster member belongs to
  - `memberName`: Name of new cluster member
  - `memberWeight`: Weight value of new cluster member
  - `memberUUID`: UUID of new cluster member
  - `genUniquePorts`: Generates unique port numbers for HTTP transports defined in the server
  - `replicatorEntry`: Enables this member to use data replication service for HTTP session persistence
2. Start the function to create the first member of PlantsCluster, *Plants1*, with the following options:

**Table 4: Options for Plants1 cluster member**

Option name	Option value
1 <code>clusterName</code>	Retrieve the value of property named "CLUSTER_NAME" from plants.properties.
2 <code>memberNode</code>	Retrieve the value of property named "PLANTS_NODE01_NAME" from plants.properties.

**Table 4: Options for Plants1 cluster member**

	<b>Option name</b>	<b>Option value</b>
3	memberName	Retrieve the value of property named "PLANTS_SERVER01_NAME" from plants.properties.
4	memberWeight	"2"
5	memberUUID	""
6	genUniquePorts	"true"
7	replicatorEntry	"false"

3. Start the function to create the second member of PlantsCluster, **plants2**, with the following options:

**Table 5: Options for plants2 cluster member**

	<b>Option name</b>	<b>Option value</b>
1	clusterName	Retrieve the value of property named "CLUSTER_NAME" from plants.properties.
2	memberNode	Retrieve the value of property named "PLANTS_NODE02_NAME" from plants.properties.
3	memberName	Retrieve the value of property named "PLANTS_SERVER02_NAME" from plants.properties.
4	memberWeight	"2"
5	memberUUID	""
6	genUniquePorts	"true"
7	replicatorEntry	"false"

## Creating the **createClusterMember** function

1. Open the **ex07\_createClusterMember.py** function skeleton script in a Jython editor view.

This file provides a function definition statement and general instructions in the form of comments to help complete the code. It also includes basic print statements that display execution progress messages.

```

1 # ex07_createClusterMember.py
2
3 def createClusterMember(clusterName,
4                         mbrNode,
5                         mbrName,
6                         mbrWeight="",
7                         mbrUUID="",
8                         genUniquePorts="true",
9                         replicatorEntry="false"):
10
11     print "Creating cluster member", mbrName, "in cluster", clusterName
12
13     #Build options list for -memberConfig step.
14     memberConfigOptions = []
15
16     # Call AdminTask to create the cluster member.
17     AdminTask.
18
19     # Save the configuration change.
20     #
21     #AdminConfig.save()
22
23     print "Cluster member", mbrName, "created in cluster", clusterName

```

2. Complete the code as instructed in the comments in lines 13 and 16. You first must identify the appropriate AdminTask command to use, understand its target, arguments, and steps, and compose its invocation syntax.



### Hint

As for the previous script, you can identify and explore the details of the AdminTask command to use by searching the Information Center for commands that belong to the *ClusterConfigCommands* command group.

After you identify the command to use, use the description and examples that are provided to figure out the target, arguments, and steps that it requires. You can also get more usage details by using one or more of the following methods:

- *AdminTask.help(aCommandName)*
- *AdminTask.help(aCommandName, aStepName)*
- *AdminTask.aCommandName("-interactive")* (runs the command in interactive mode)

**createClusterMember**

The `createClusterMember` command creates a member of a server cluster. A cluster member is an application server that belongs to a cluster. If this is the first member of the cluster, you must specify a template to use as the model for the cluster member. The template can be either a default server template, or an existing application server.

- Using Jython list:

First member creation using template name:

```
AdminTask.createClusterMember(['-clusterName', 'cluster1',
'-memberConfig',
'[-memberNode node1 -memberName member1 -genUniquePorts true
-replicatorEntry false]]',
'-firstMember', '[-templateName serverTemplateName]'])
```

First member creation using server and node for template:

```
AdminTask.createClusterMember(['-clusterName', 'cluster1',
'-memberConfig', '[-memberNode
node1 -memberName member1 -genUniquePorts true -replicatorEntry
false]', '-firstMember',
'[-templateServerNode node1 -templateServerName server1]'])
```

Second member creation:

```
AdminTask.createClusterMember(['-clusterName', 'cluster1',
'-memberConfig', '[-memberNode
node1 -memberName member1 -genUniquePorts true -replicatorEntry
false]'])
```

Your completed *createClusterMember* function should look as follows:

```


  1 # ex07_createClusterMember.py
  2
  3 def createClusterMember(clusterName,
  4                         mbrNode,
  5                         mbrName,
  6                         mbrWeight="",
  7                         mbrUUID="",
  8                         genUniquePorts="true",
  9                         replicatorEntry="false"):
 10
 11     print "Creating cluster member", mbrName, "in cluster", clusterName
 12
 13     #Build options list for -memberConfig step.
 14     memberConfigOptions = [mbrNode, mbrName, mbrWeight, mbrUUID, genUniquePorts, replicatorEntry]
 15
 16     # Call AdminTask to create the cluster member.
 17     AdminTask.createClusterMember(["-clusterName", clusterName, "-memberConfig", [memberConfigOptions]])
 18
 19     # Save the configuration change.
 20     #
 21     #AdminConfig.save()
 22
 23     print "Cluster member", mbrName, "created in cluster", clusterName

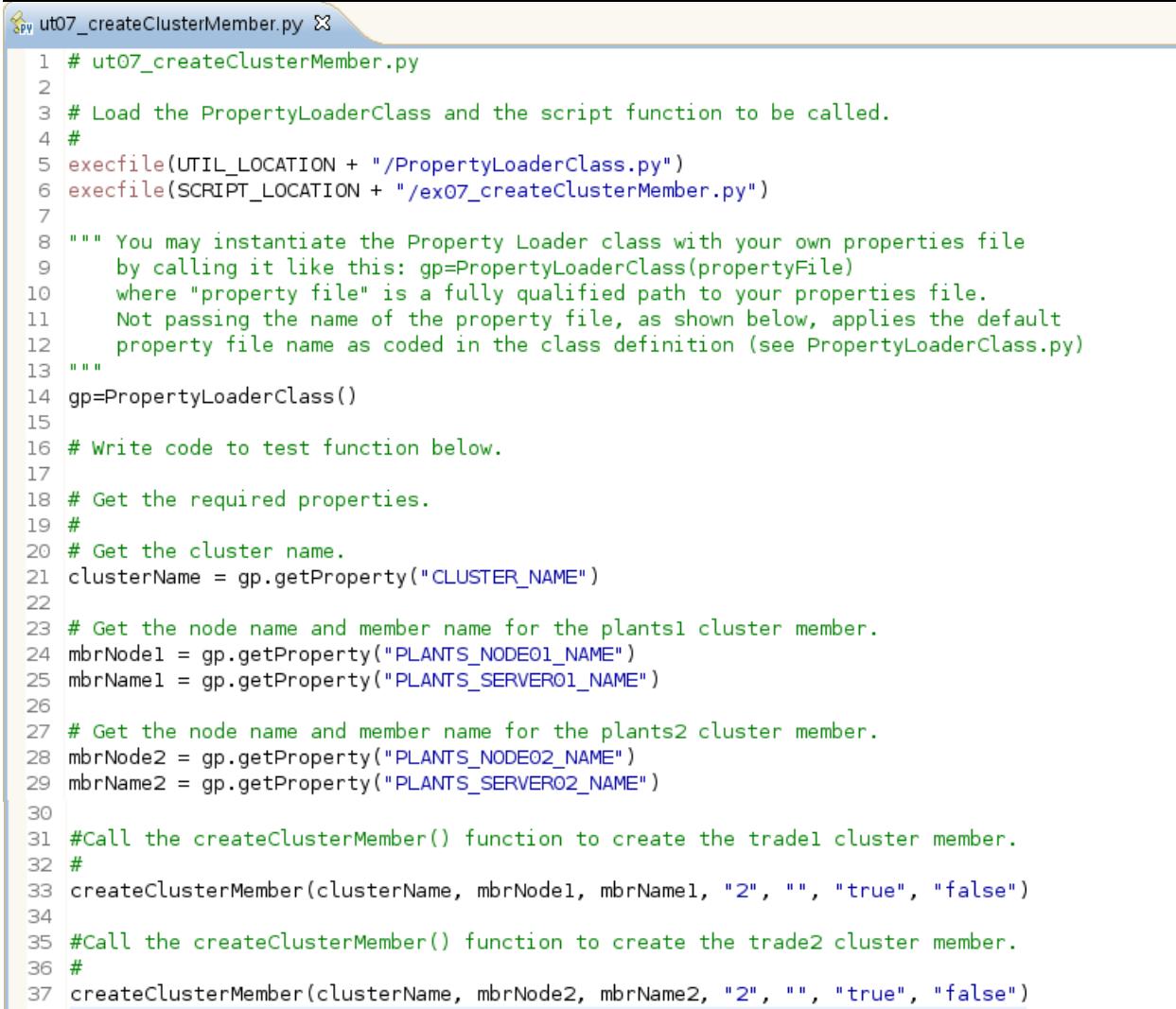
```

- \_\_\_ 3. Save your changes.

## Starting the *createClusterMember* function

- \_\_\_ 1. Use the *ut07\_createClusterMember.py* script to start the *createClusterMember* function that you created. Modify it to:
  - \_\_\_ a. Retrieve any properties that are required to create the *Plants1* and *Plants2* cluster members as listed in the options table, by using the *gp.getProperty(propertyName)* method.
  - \_\_\_ b. Start the *createClusterMember* function to create the *Plants1* cluster member with the parameter values specified in the options table.
  - \_\_\_ c. Start the *createClusterMember* function to create the *Plants2* cluster member with the parameter values specified in the options table.

Your ut07\_createClusterMember.py script should look as follows:



```

ut07_createClusterMember.py ✘
1 # ut07_createClusterMember.py
2
3 # Load the PropertyLoaderClass and the script function to be called.
4 #
5 execfile(UTIL_LOCATION + "/PropertyLoaderClass.py")
6 execfile(SCRIPT_LOCATION + "/ex07_createClusterMember.py")
7
8 """ You may instantiate the Property Loader class with your own properties file
9     by calling it like this: gp=PropertyLoaderClass(propertyFile)
10    where "property file" is a fully qualified path to your properties file.
11    Not passing the name of the property file, as shown below, applies the default
12    property file name as coded in the class definition (see PropertyLoaderClass.py)
13 """
14 gp=PropertyLoaderClass()
15
16 # Write code to test function below.
17
18 # Get the required properties.
19 #
20 # Get the cluster name.
21 clusterName = gp.getProperty("CLUSTER_NAME")
22
23 # Get the node name and member name for the plants1 cluster member.
24 mbrNode1 = gp.getProperty("PLANTS_NODE01_NAME")
25 mbrName1 = gp.getProperty("PLANTS_SERVER01_NAME")
26
27 # Get the node name and member name for the plants2 cluster member.
28 mbrNode2 = gp.getProperty("PLANTS_NODE02_NAME")
29 mbrName2 = gp.getProperty("PLANTS_SERVER02_NAME")
30
31 #Call the createClusterMember() function to create the tradel cluster member.
32 #
33 createClusterMember(clusterName, mbrNode1, mbrName1, "2", "", "true", "false")
34
35 #Call the createClusterMember() function to create the trade2 cluster member.
36 #
37 createClusterMember(clusterName, mbrNode2, mbrName2, "2", "", "true", "false")

```

- \_\_ 2. Save ut07\_createClusterMember.py.
- \_\_ 3. Run **ut07\_createClusterMember.py** from a terminal window.
  - \_\_ a. Go to <profile\_root>/Dmgr/bin.
  - \_\_ b. Enter the following command all on one line or copy it from the **/usr/Solutions/Exercise07/cmd.txt** file:
 

```
./wsadmin.sh -user wasadmin -password webSphere -profile /usr/LabWork/DistributedWS/PlantsScriptingProject/Utilities /setScriptEnvironment.py -f /usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut07_createClusterMember.py
```
  - \_\_ c. Examine the output from this command in the terminal window. Correct any errors that might show up and rerun until successful.

- \_\_\_ d. When you are satisfied that your script works, edit `ex07_createClusterMember.py` to uncomment the `AdminConfig.save()` line; then save and unit test it again by running `ut07_createCluster.py` from a terminal window.
- \_\_\_ e. When you are satisfied that your script works, edit `ex07_createClusterMember.py` to uncomment the `AdminConfig.save()` line; then save and unit test it again by running `ut07_createCluster.py` from a terminal window.



### Important

Make sure to run the script again with the `AdminConfig.save()` line un-commented; otherwise, you do not see the effect of your changes when you verify the results in the next section.

## Verifying the results

- \_\_\_ 1. Using the administrative console, verify that the two cluster members were created.
  - \_\_\_ a. Start the Firefox web browser, and enter the web address  
`http://was85host:9061/ibm/console`
  - \_\_\_ b. Log in using `wasadmin` as the user ID and `web1sphere` as the password.
  - \_\_\_ c. In the navigator pane, select **Servers > Clusters > WebSphere application server clusters**. The right pane shows the *PlantsCluster* that you created in the previous step.

Select	Name	Status
<input type="checkbox"/>	PlantsCluster	

- \_\_\_ d. Click the **PlantsCluster** link.

- \_\_\_ e. Under Additional Properties, click the + button beside the Cluster members link.

The screenshot shows the 'WebSphere application server clusters' interface. In the top navigation bar, the 'Local Topology' tab is selected. On the left, under 'General Properties', there are fields for 'Cluster name' (PlantsCluster) and 'Bounding node group name' (DefaultNodeGroup). Under 'Additional Properties', the 'Cluster members' link is highlighted with a red box. A table below it lists two cluster members: Plants1 and Plants2, each with a weight of 2. Other links in this section include 'Messaging engines', 'Communications', 'Backup cluster', 'Dynamic workload management (DWLM)', 'Endpoint listeners', 'Security domain', and 'Web server plug-in cluster properties'.

Cluster member	Weight
Plants1	2
Plants2	2

The two cluster members, **Plants1** and **Plants2**, are shown with the correct weight value of 2.

- \_\_\_ f. Click the **Cluster Members** link.

The screenshot shows the 'Cluster members' configuration interface. At the top, there are buttons for 'New...', 'Delete', 'Templates...', 'Start', 'Stop', 'Restart', 'ImmediateStop', 'Terminate', and 'Make Idle'. Below this is a search bar with dropdowns for 'Member name', 'Node', 'Host Name', 'Version', 'Configured weight' (with an 'Update' button), 'Runtime weight' (with an 'Update' button), and 'Status'. A table lists the cluster members: Plants1 and Plants2. Each row includes a checkbox, the member name, host name, version, configured weight, runtime weight, and status. The total count at the bottom is 2.

Select	Member name	Node	Host Name	Version	Configured weight	Runtime weight	Status
<input type="checkbox"/>	<a href="#">Plants1</a>	PlantsNode01	was85host	ND 8.5.0.0	<input type="text" value="2"/>	<input type="text"/>	<a href="#">?</a>
<input type="checkbox"/>	<a href="#">Plants2</a>	PlantsNode02	was85host	ND 8.5.0.0	<input type="text" value="2"/>	<input type="text"/>	<a href="#">?</a>

This pane shows more configuration detail for each cluster member. Confirm that the *Plants1* and *Plants2* cluster members are on the *PlantsNode01* and *PlantsNode02* nodes.

- \_\_\_ g. Log out of the administrative console.
- \_\_\_ 2. Close all open editor views.

You successfully developed a Jython script to create a cluster member.

## Section 7: Developing scripts to query the state of a cluster, start, or stop a cluster

In this step, you create parameterized Jython functions to perform the common administration tasks of querying the state of a cluster, starting a cluster and stopping a cluster. Since these tasks are operational tasks, you use the AdminControl object to implement these functions.

### Requirements specification

Your specific requirements are as follows:

1. Create a Jython function named `queryClusterState` to display the state of a cluster whose name is passed as a parameter.
2. Test the function on `PlantsCluster` by using the provided `ut07_queryClusterState.py` unit test script.
3. Create a Jython function named `startCluster` to start a cluster whose name is passed as a parameter.
4. Test the function on `PlantsCluster` by using the provided `ut07_startCluster.py` unit test script.
5. Create a Jython function named `stopCluster` to stop a cluster whose name is passed as a parameter.
6. Test the function on `PlantsCluster` by using the provided `ut07_stopCluster.py` unit test script.

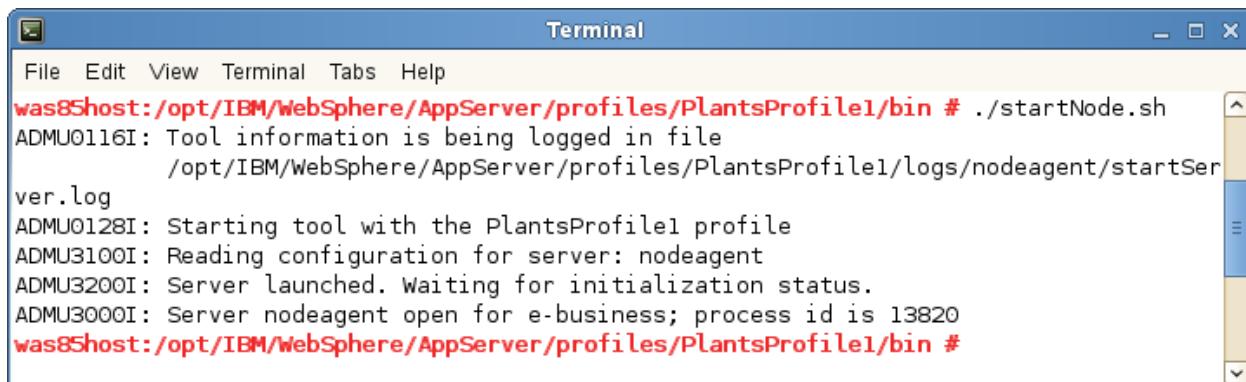
### Starting the node agents

Begin by starting the `PlantsProfile1` and `PlantsProfile2` node agents. The node agents must be started because starting a cluster starts the server process of each cluster member by calling its associated node agent. If the node agents for the `Plants1` and `Plants2` servers are not running, the `PlantsCluster` cannot start.

- \_\_\_ 1. Start the node agent on **PlantsProfile1**.
  - \_\_\_ a. Open a terminal window and go to the `<profile_root>/PlantsProfile1/bin` directory.

- \_\_\_ b. Run the following command:

```
./startNode.sh
```



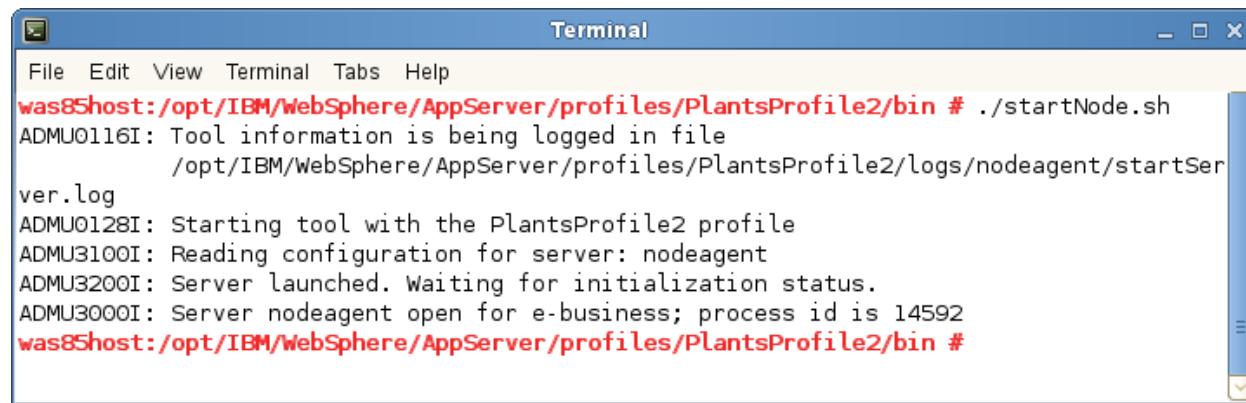
```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/PlantsProfile1/bin # ./startNode.sh
ADMU0116I: Tool information is being logged in file
            /opt/IBM/WebSphere/AppServer/profiles/PlantsProfile1/logs/nodeagent/startSer
ver.log
ADMU0128I: Starting tool with the PlantsProfile1 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server nodeagent open for e-business; process id is 13820
was85host:/opt/IBM/WebSphere/AppServer/profiles/PlantsProfile1/bin #
```

- \_\_\_ c. Wait until you see the message “Server nodeagent open for e-business; process id is \_\_\_\_\_. Copy your process id on the blank line for future reference.

\_\_\_ 2. Start the node agent on **PlantsProfile2**.

- \_\_\_ a. In the terminal window, go to the <*profile\_root*>\PlantsProfile2\bin directory.
- \_\_\_ b. Run the following command:

```
./startNode.sh
```



```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/PlantsProfile2/bin # ./startNode.sh
ADMU0116I: Tool information is being logged in file
            /opt/IBM/WebSphere/AppServer/profiles/PlantsProfile2/logs/nodeagent/startSer
ver.log
ADMU0128I: Starting tool with the PlantsProfile2 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server nodeagent open for e-business; process id is 14592
was85host:/opt/IBM/WebSphere/AppServer/profiles/PlantsProfile2/bin #
```

- \_\_\_ c. Wait until you see the message “Server nodeagent open for e-business; process id is \_\_\_\_\_. Copy your process id on the blank line for future reference.
- \_\_\_ d. Leave the terminal window **opened** or **minimized**. You are going to use it later to stop the node agents.



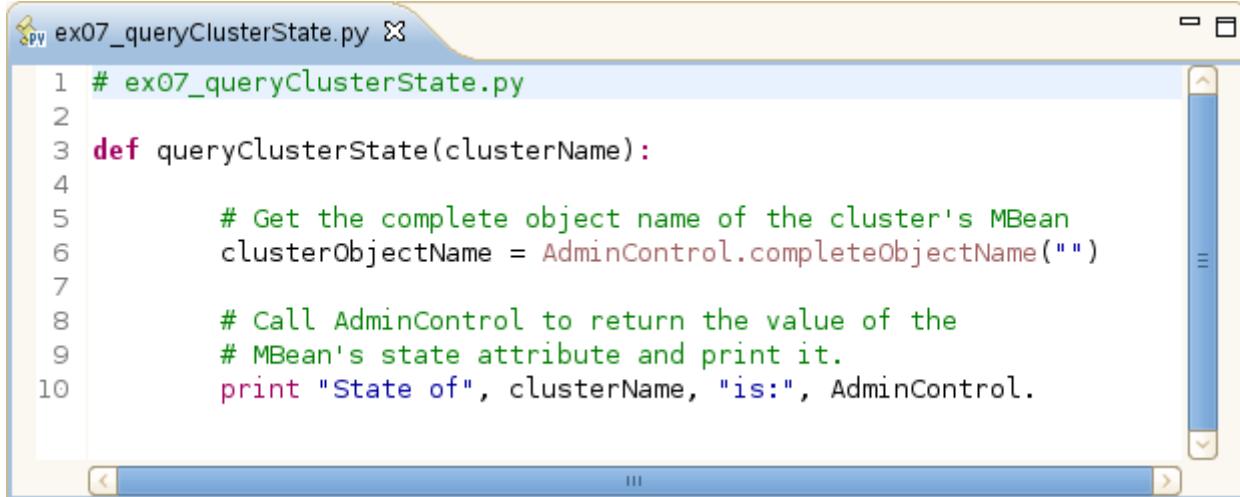
### Information

As in the administrative console, the node agents cannot be started by using wsadmin.

## Creating the queryClusterState function

1. Open the **ex07\_queryClusterState.py** function skeleton script in a Jython editor view.

This file provides a function definition statement and general instructions in the form of comments to help complete the code.



The screenshot shows a Jython editor window titled "ex07\_queryClusterState.py". The code is as follows:

```
1 # ex07_queryClusterState.py
2
3 def queryClusterState(clusterName):
4
5     # Get the complete object name of the cluster's MBean
6     clusterObjectName = AdminControl.completeObjectName("")
7
8     # Call AdminControl to return the value of the
9     # MBean's state attribute and print it.
10    print "State of", clusterName, "is:", AdminControl.
```

2. Complete the code as instructed in the comments in lines 5 and 8. Use the Information Center and other available resources to determine the object name template and AdminControl method to use.

**Hint**

Search the Information Center for the topic named *Clustering servers with scripting*. It provides a good starting point for finding more about writing scripts to query, start, or stop clusters. For help on using a particular method, remember that you can use `AdminControl.help(aMethodName)`.

[Network Deployment \(All operating systems\), Version 8.5](#) > [Scripting the application serving environment \(wsadmin\)](#)

## Clustering servers with wsadmin scripting

You can use **scripting** and the **wsadmin** tool to **cluster** application **servers**, generic **servers**, web **servers**, and proxy **servers**.

### About this task

This topic contains the following tasks:

### Procedure

- Creating **clusters** using **scripting**
- Modifying **cluster** member templates using **wsadmin scripting**
- Creating **cluster** members using **scripting**
- Creating **clusters** without **cluster** members using **scripting**
- Starting **clusters** using **scripting**
- Querying **cluster** state using **scripting**
- Stopping **clusters** using **scripting**

[Network Deployment \(All operating systems\), Version 8.5](#) > [Scripting the application serving environment \(wsadmin\)](#) > [Clustering servers with wsadmin scripting](#)

## Querying cluster state using scripting

You can query cluster states using the wsadmin tool and scripting.

### Before you begin

Before starting this task, the wsadmin tool must be running. See the topic about starting the wsadmin scripting client using wsadmin scripting for more information.

### About this task

Perform the following steps to query cluster state:

#### Procedure

1. Identify the Cluster MBean and assign it to the cluster variable.

- o Using JACL:

```
set cluster [$AdminControl  
completeObjectName  
cell=mycell,type=Cluster,name=cluster1,*]
```

- o Using Jython:

```
cluster =  
AdminControl.completeObjectName('cell=mycel  
l,type=Cluster,name=cluster1,*')  
print cluster
```

This command returns the Cluster MBean.

Example output:

```
WebSphere:cell=mycell,name=cluster1,mbeanIdentif  
ier=Cluster,type=Cluster,process=cluster1
```

2. Query the cluster state.

- o Using JACL:

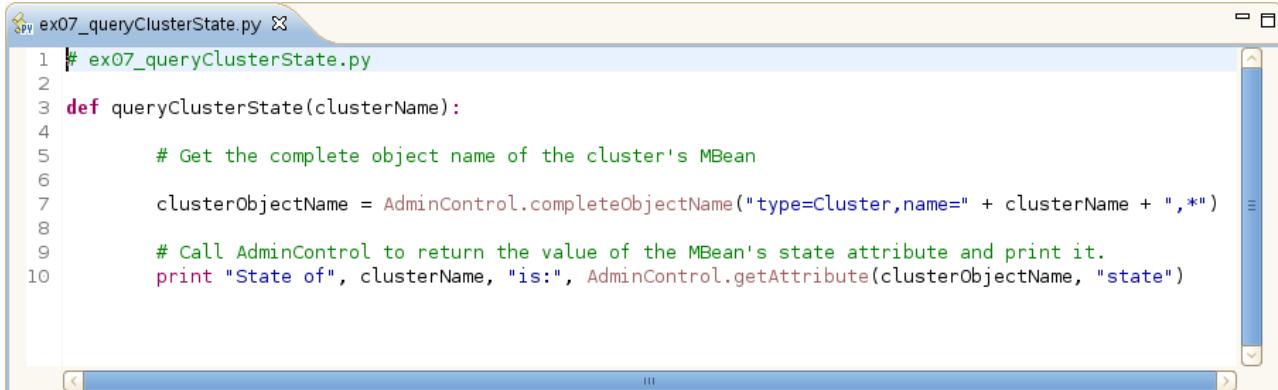
```
$AdminControl getAttribute $cluster state
```

- o Using Jython:

```
AdminControl.getAttribute(cluster, 'state')
```

This command returns the value of the run-time state attribute.

Your completed *queryClusterState* function should look as follows:



```

 1 # ex07_queryClusterState.py
 2
 3 def queryClusterState(clusterName):
 4
 5     # Get the complete object name of the cluster's MBean
 6
 7     clusterObjectName = AdminControl.completeObjectName("type=Cluster,name=" + clusterName + ",*")
 8
 9     # Call AdminControl to return the value of the MBean's state attribute and print it.
10     print "State of", clusterName, "is:", AdminControl.getAttribute(clusterObjectName, "state")

```

- \_\_\_ 3. Save your changes.

## Starting the *queryClusterState* function

- \_\_\_ 1. The *ut07\_queryClusterState.py* script is completed for you and is provided to unit test the *queryClusterState* function that you created. It retrieves the name of the *PlantsCluster* from the *plants.properties* file and starts the *queryClusterState* function. You can open it in a Jython editor and review its code if you like.
- \_\_\_ 2. Run ***ut07\_queryClusterState.py*** from a terminal window.

- \_\_\_ a. Enter the following command all on one line or copy it from the **/usr/Solutions/Exercise07/cmd.txt** file:

```

./wsadmin.sh -user wasadmin -password websphere -profile
/usr/LabWork/DistributedWS/PlantsScriptingProject/Utilities
/setScriptEnvironment.py -f
/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/u
t07_queryClusterState.py

```

- \_\_\_ b. Examine the output from this command in the terminal window.
- \_\_\_ c. If there are no errors in your script, you should see output as follows: State of PlantsCluster is: websphere.cluster.stopped

The message indicates that the *PlantsCluster* is stopped, as expected.

## Simplifying the commands

In this part, you apply what you learned earlier in the course to simplify running the complex wsadmin commands. Recall that user ID and password data can be stored in the *soap.client.props* file. Also, remember that profile scripts can be configured to run automatically when wsadmin starts by editing the *wsadmin.properties* file. Finally,

recall that within the wsadmin interactive shell, the `execfile()` command can be used to run scripts.

- \_\_\_ 1. Store the administrator user ID and password data in the `soap.client.props` file.

- \_\_\_ a. From a terminal window, go to `<profile_root>/Dmgr/properties`.

- \_\_\_ b. Use a text editor to open the `soap.client.props` file.

- \_\_\_ c. Enter the user ID and password data as follows:

```
com.ibm.SOAP.loginUserId=wasadmin
```

```
com.ibm.SOAP.loginPassword=web1sphere
```

- \_\_\_ d. Save the changes and close the file.

- \_\_\_ e. Remember to run the `PropFilePasswordEncoder` utility to encode the password.

- \_\_\_ 2. Modify the `wsadmin.properties` file to use Jython and the script environment profile.

- \_\_\_ a. Use a text editor to open the `wsadmin.properties` file.

- \_\_\_ b. Make Jython the default scripting language by modifying the following line.

```
com.ibm.ws.scripting.defaultLang=jython
```

- \_\_\_ c. Use the Jython versions of the existing scripting profile by changing their file extensions to `.py` as follows.

```
com.ibm.ws.scripting.profiles=/opt/IBM/WebSphere/AppServer/  
bin/securityProcs.py;/opt/IBM/WebSphere/AppServer/bin/LTPA_  
LDAPSecurityProcs.py
```

- \_\_\_ d. Add the scripting environment profile as follows.

```
com.ibm.ws.scripting.profiles=/opt/IBM/WebSphere/AppServer/  
bin/securityProcs.py;/opt/IBM/WebSphere/AppServer/bin/LTPA_  
LDAPSecurityProcs.py;/usr/LabWork/DistributedWS/PlantsScrip  
tingProject/Utilities/setScriptEnvironment.py
```

- \_\_\_ e. Save the changes and close the file.

- \_\_\_ 3. Run scripts within the wsadmin shell by using the `execfile()` command.

- \_\_\_ a. From a terminal window, go to `<profile_root>/Dmgr/bin`

- \_\_\_ b. Start the wsadmin interactive shell by entering: `./wsadmin.sh`

- \_\_\_ c. Try checking the cluster status again by entering the following command:

```
execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject
/Scripts/ut07_queryClusterState.py")
```

```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./wsadmin.sh
WASX7209I: Connected to process "dmgr" on node dmgrNode using SOAP connector;
The type of process is: DeploymentManager
WASX7031I: For help, enter: "print Help.help()"
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut07_queryClusterState.py")
State of PlantsCluster is: websphere.cluster.stopped
wsadmin>
```

- \_\_\_ d. Now you need use only the `execfile` command to run the Jython scripts. Keep the wsadmin shell open and use it to run the remaining scripts in this exercise.

## Creating the startCluster function

- \_\_\_ 1. Open the `ex07_startCluster.py` function skeleton script in a Jython editor view.

This file provides a function definition statement and general instructions in the form of comments to help complete the code.

```
ex07_startCluster.py
1 # ex08_startCluster.py
2
3 def startCluster(clusterName):
4
5     print "Starting cluster", clusterName
6
7     # Get the complete object name of the cluster's MBean
8     clusterObjectName = AdminControl.completeObjectName("type=Cluster,name=" + clusterName + ",*")
9
10    # Call AdminControl to start the cluster.
11    AdminControl.
12
13    print "Cluster", clusterName, "started"
```

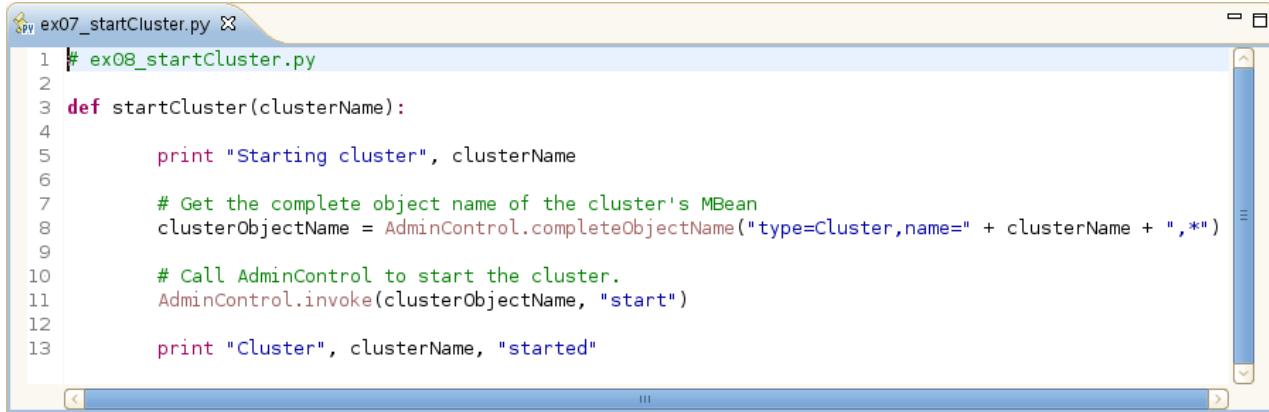
- \_\_\_ 2. Complete the code as instructed in the comment in line 10. Use the Information Center and other available resources to determine the AdminControl method to use.



### Hint

The Information Center topic named *Clustering servers with scripting* provides a good starting point for finding more about writing scripts to query, start, or stop clusters. For help on using a particular method, remember that you can use `AdminControl.help(aMethodName)`.

Your completed *startCluster* function should look as follows:



```

1 # ex07_startCluster.py
2
3 def startCluster(clusterName):
4
5     print "Starting cluster", clusterName
6
7     # Get the complete object name of the cluster's MBean
8     clusterObjectName = AdminControl.completeObjectName("type=Cluster,name=" + clusterName + ",*")
9
10    # Call AdminControl to start the cluster.
11    AdminControl.invoke(clusterObjectName, "start")
12
13    print "Cluster", clusterName, "started"

```

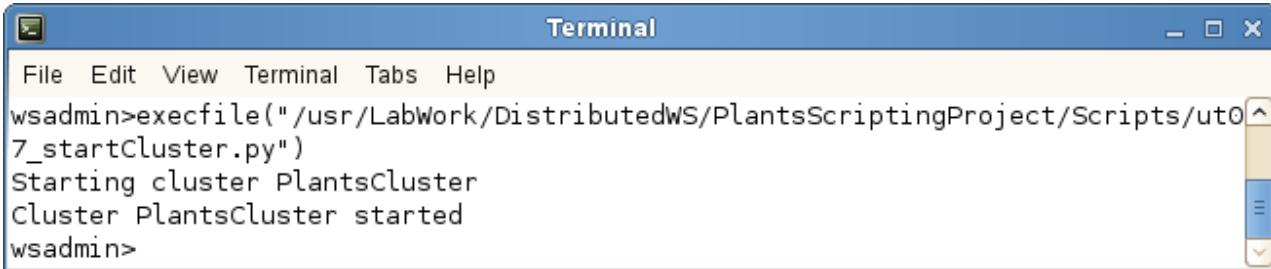
- \_\_\_ 3. Save your changes.

## Starting the *startCluster* function

- \_\_\_ 1. The *ut07\_startCluster.py* script is completed for you and is provided to unit test the *startCluster* function that you created. It retrieves the name of the *PlantsCluster* from the *plants.properties* file and starts the *startCluster* function. You can open it in a Jython editor and review its code if you like.
- \_\_\_ 2. Run ***ut07\_startCluster.py*** from a terminal window.

- \_\_\_ a. Enter the following command in the wsadmin shell:

```
execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject
/Scripts/ut07_startCluster.py")
```



```

File Edit View Terminal Tabs Help
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/uto
7_startCluster.py")
Starting cluster PlantsCluster
Cluster PlantsCluster started
wsadmin>

```

- \_\_\_ b. Examine the output from this command in the terminal window.
- \_\_\_ c. If there are no errors in your script, you should see output as follows:

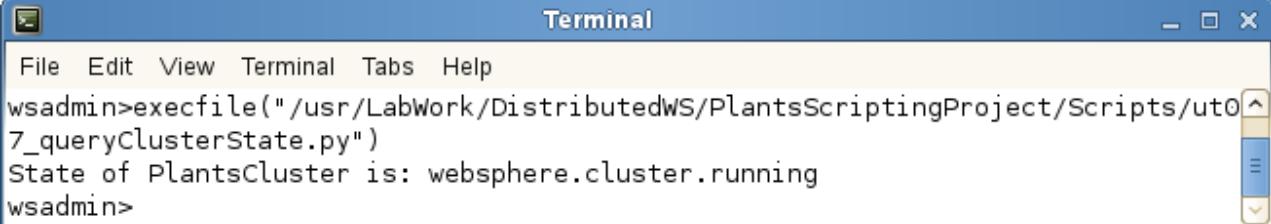
```
Starting cluster PlantsCluster
Cluster PlantsCluster started
```



### Note

Although the last message, indicating that the *PlantsCluster* is started, is displayed after only a few seconds, the process of starting all of its members takes a few minutes.

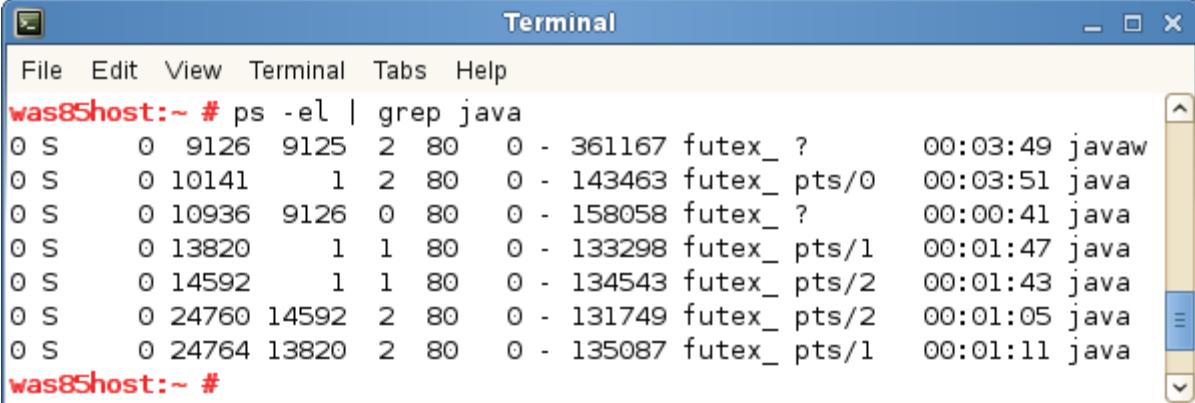
- \_\_\_ 3. Verify that the *PlantsCluster* is started.
- \_\_\_ a. Run `ut07_queryClusterState.py` in the terminal window to query the state of *PlantsCluster*. If the start is still in progress, you get a message that indicates that the state is partially started as follows:
- State of PlantCluster is: websphere.cluster.partial.start.
- \_\_\_ b. Wait a few minutes and query the cluster state again. When it is started, its state is as follows:



```
Terminal
File Edit View Terminal Tabs Help
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut07_queryClusterState.py")
State of PlantsCluster is: websphere.cluster.running
wsadmin>
```

- \_\_\_ c. Both the *Plants1* and *Plants2* cluster member application servers are started. Use the system `ps` command to verify that there are 5 `java` processes running that represent the following processes:
- Deployment manager
  - PlantsProfile1 nodeagent
  - PlantsProfile2 nodeagent
  - Plants1 cluster member
  - Plants2 cluster member

Open a terminal window, and enter the command: `ps -el | grep java`



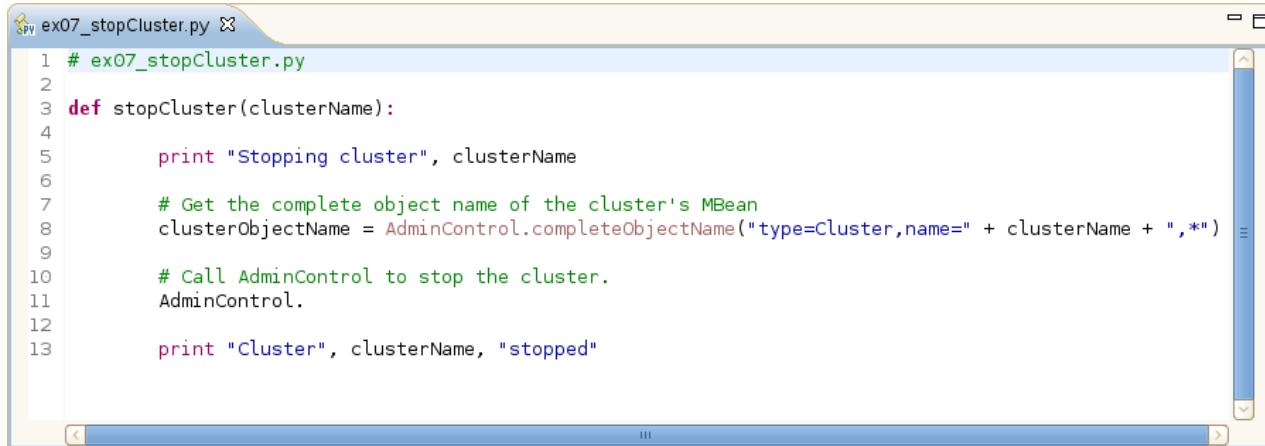
```
Terminal
File Edit View Terminal Tabs Help
was85host:~ # ps -el | grep java
0 S 0 9126 9125 2 80 0 - 361167 futex_ ? 00:03:49 javaw
0 S 0 10141 1 2 80 0 - 143463 futex_ pts/0 00:03:51 java
0 S 0 10936 9126 0 80 0 - 158058 futex_ ? 00:00:41 java
0 S 0 13820 1 1 80 0 - 133298 futex_ pts/1 00:01:47 java
0 S 0 14592 1 1 80 0 - 134543 futex_ pts/2 00:01:43 java
0 S 0 24760 14592 2 80 0 - 131749 futex_ pts/2 00:01:05 java
0 S 0 24764 13820 2 80 0 - 135087 futex_ pts/1 00:01:11 java
was85host:~ #
```

You can identify the Java processes of your node agents by their process IDs. Notice that the node agent process is the parent of the cluster member process.

## Creating the `stopCluster` function

- \_\_\_ 1. Open the `ex07_stopCluster.py` function skeleton script in a Jython editor view.

This file provides a function definition statement and general instructions in the form of comments to help complete the code.



```

SPV ex07_stopCluster.py ☰
1 # ex07_stopCluster.py
2
3 def stopCluster(clusterName):
4
5     print "Stopping cluster", clusterName
6
7     # Get the complete object name of the cluster's MBean
8     clusterObjectName = AdminControl.completeObjectName("type=Cluster,name=" + clusterName + ",*")
9
10    # Call AdminControl to stop the cluster.
11    AdminControl.
12
13    print "Cluster", clusterName, "stopped"

```

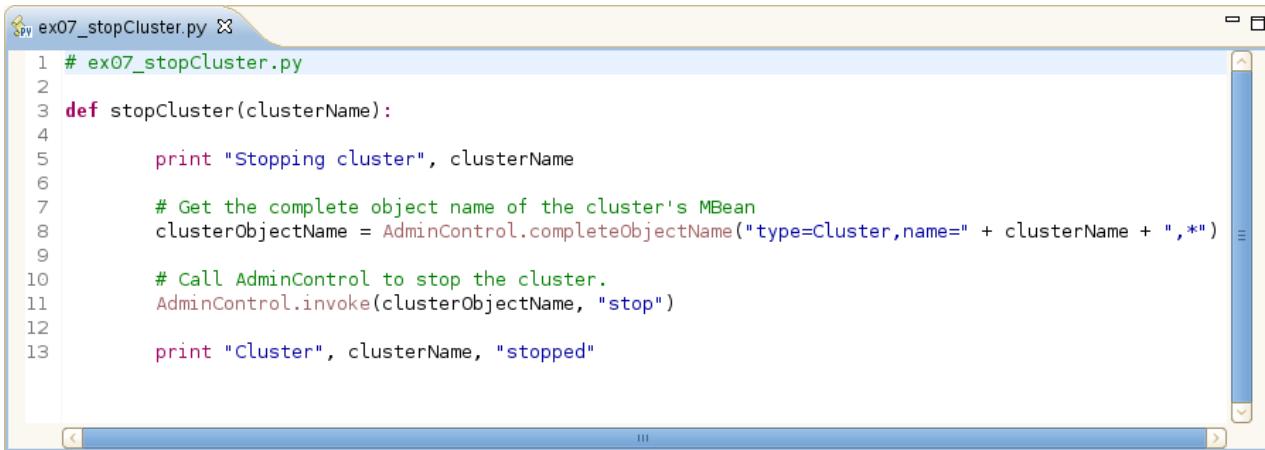
- 2. Complete the code as instructed in the comment in line 10. Use the Information Center and other available resources to determine the AdminControl method to use.



### Hint

The Information Center topic named *Clustering servers with scripting* provides a good starting point for finding more about writing scripts to query, start, or stop clusters. For help on using a particular method, remember that you can use `AdminControl.help(aMethodName)`.

Your completed `stopCluster` function should look as follows:



```

SPV ex07_stopCluster.py ☰
1 # ex07_stopCluster.py
2
3 def stopCluster(clusterName):
4
5     print "Stopping cluster", clusterName
6
7     # Get the complete object name of the cluster's MBean
8     clusterObjectName = AdminControl.completeObjectName("type=Cluster,name=" + clusterName + ",*")
9
10    # Call AdminControl to stop the cluster.
11    AdminControl.invoke(clusterObjectName, "stop")
12
13    print "Cluster", clusterName, "stopped"

```

- 3. Save your changes.

## Starting the `stopCluster` function

- 1. The `ut07_stopCluster.py` script is completed for you and is provided to unit test the `stopCluster` function that you created. It retrieves the name of the *PlantsCluster* from

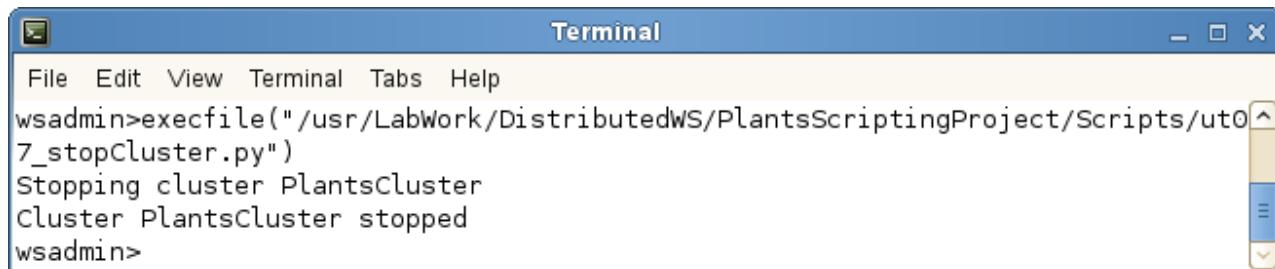
the *plants.properties* file and starts the *stopCluster* function. You can open it in a Jython editor and review its code if you like.

- \_\_\_ 2. Run **ut07\_stopCluster.py** from a terminal window.

- \_\_\_ a. Enter the following command in the wsadmin shell:

```
execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/
/Scripts/ ut07_stopCluster.py")
```

- \_\_\_ b. If there are no errors in your script, you should see output as follows:



```
File Edit View Terminal Tabs Help
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ ut07_stopCluster.py")
Stopping cluster PlantsCluster
Cluster PlantsCluster stopped
wsadmin>
```



### Note

As was the case with starting the cluster, the message that indicates that the *PlantsCluster* is stopped, is displayed after only a few seconds. However, the process of stopping all of its members takes a few minutes.

- \_\_\_ 3. Verify that the *PlantsCluster* is stopped.

- \_\_\_ a. Run **ut07\_queryClusterState.py** in the terminal window to query the state of *PlantsCluster*. If the stop is still in progress, you get a message that indicates that the state is “partial.stop”.
  - \_\_\_ b. Wait 2 or 3 minutes and query the cluster state again. When it is stopped, its state is “stopped”.
  - \_\_\_ c. Run the **ps -el | grep java** command again to verify that the Java processes for the cluster members are no longer running.

You successfully created Jython scripts to query the state of a cluster, start a cluster and stop a cluster.

## Section 8: Cleaning up the environment

- \_\_\_ 1. Stop the wsadmin shell by entering: quit.
- \_\_\_ 2. Stop the deployment manager server.
  - \_\_\_ a. From the terminal window, stop the deployment manager by entering the following command:  
`./stopManager.sh -username wasadmin -password web1sphere`
- \_\_\_ 3. Close any scripts that are still open in the Jython editor.
- \_\_\_ 4. Exit the IBM Assembly and Deploy Tools by selecting **File > Exit**. The workspace is automatically saved.
- \_\_\_ 5. Stop the node agent on **PlantsProfile2**.
  - \_\_\_ a. In the terminal window for `<profile_root>/PlantsProfile2/bin` (it should still be opened from when you started the node agent), run the following command:  
`./stopNode.sh -user wasadmin -password web1sphere`
  - \_\_\_ b. Wait until you see the message “Server nodeagent stop completed”.
- \_\_\_ 6. Stop the node agent on **PlantsProfile1**.
  - \_\_\_ a. In the Command Prompt window, go to `<profile_root>/PlantsProfile1/bin` and run the following command:  
`./stopNode.sh -user wasadmin -password web1sphere`
  - \_\_\_ b. Wait until you see the message “Server nodeagent stop completed”.

## End of exercise

## Exercise review and wrap-up

In this exercise, you created the server and server resources that the PlantsByWebSphere application requires by using silent installations and scripting. Specifically, you:

- Created a deployment manager and a managed node profile by using a response file.
- Developed Jython scripts to query the state of a cluster, and start or stop a cluster.

You also learned good script design and unit testing practices, and produced flexible and reusable scripts.



# Exercise 8. Installing and configuring the IBM HTTP Server

## What this exercise is about

In this exercise, you use the IBM Installation Manager (IIM) to record a response file that you use to silently install the IBM HTTP Server and its plug-in. You then develop Jython administrative scripts to configure the IBM HTTP Server as an unmanaged node.

## What you should be able to do

At the end of this exercise, you should be able to:

- Install the IBM HTTP Server and its plug-in by using a silent installation
- Develop Jython scripts to configure the IBM HTTP Server as an unmanaged node

## Introduction

The deployment environment for the PlantsByWebSphere application is based on a distributed server topology and consists of:

- A deployment manager node
- Two managed nodes that each contain an PlantsByWebSphere application server
- A cluster that contains the two application servers as cluster members
- An unmanaged IBM HTTP Server node

Figure 1 highlights the servers and server resources that you create and configure in this exercise.

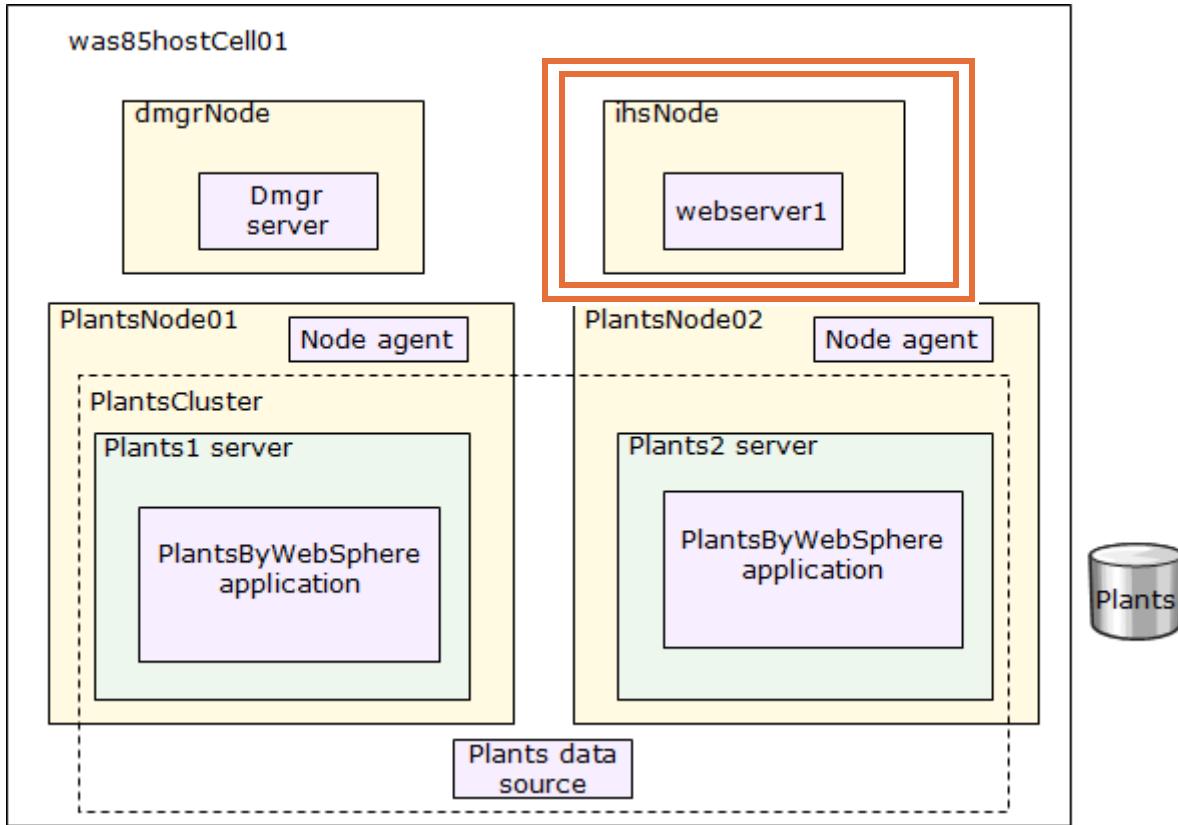


Figure 1 - Plants deployment environment servers and server resources

In the previous exercise, you created most of the Plants server environment. In this exercise, you complete the remaining steps as outlined.

1. Perform a silent installation of the IBM HTTP Server (IHS) and its plug-in.
2. Configure the Web Server plug-in.
3. Create an unmanaged node and web server definition by using scripting.

## Requirements

To complete this exercise, you must have the IBM Installation Manager and the WebSphere Application Server Network Deployment V8.5 product installed. IBM Assembly and Deploy Tools for WebSphere Administration V8.5 is also required. The installation of WebSphere Customization Toolbox 8.5.0.0 is required before beginning this exercise. The WCT is installed on the lab image. Also, a local

repository that contains the IBM software products must be created. This repository is created on the lab image and is in the file system at **/usr/IBM-repositories**.

## Exercise instructions



### Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

**<was\_root>**: /opt/IBM/WebSphere/AppServer

**<profile\_root>**: /opt/IBM/WebSphere/AppServer/profiles

### ***Section 1: Installing the IBM HTTP Server and its plug-in silently***

In this section, you perform a silent installation of the IBM HTTP Server (IHS) and its plug-in. As with other silent installations, you begin by building a response file to contain all of the wanted installation options. You use the IBM Installation Manager to record a response file for installing the IBM HTTP Server and Web Server Plug-ins. You then start the installer program by passing it the response file, and finally verify the results.

- 1. Use the IBM Installation Manager to record a response file for installing IBM HTTP Server for WebSphere Application Server and the Web Server Plug-in



## Information

To find out how to run the IBM Installation Manager in record mode for installing the IBM HTTP Server, search the WebSphere Application Server Information Center for: *Installing IBM HTTP Server silently*.

### About this task

Complete this procedure to **install IBM HTTP Server silently**.

### Procedure

1. **Record a response file to install IBM HTTP Server:** On one of your systems, complete the following actions to record a response file that will **install IBM HTTP Server**.
  - a. From a command line, change to the `eclipse` subdirectory in the directory where you **installed Installation Manager**.
  - b. Start **Installation Manager** from the command line using the `-record` option.

For example:

- **Windows Administrator or non-administrator:**

```
IBMMIM.exe -skipInstall "C:\temp\imRegistry" -record C:\temp\install_response_file.xml
```

- **AIX | HP-UX | Linux | Solaris Administrator:**

```
./IBMMIM -skipInstall /var/temp/imRegistry -record /var/temp/install_response_file.xml
```

- **AIX | HP-UX | Linux | Solaris Non-administrator:**

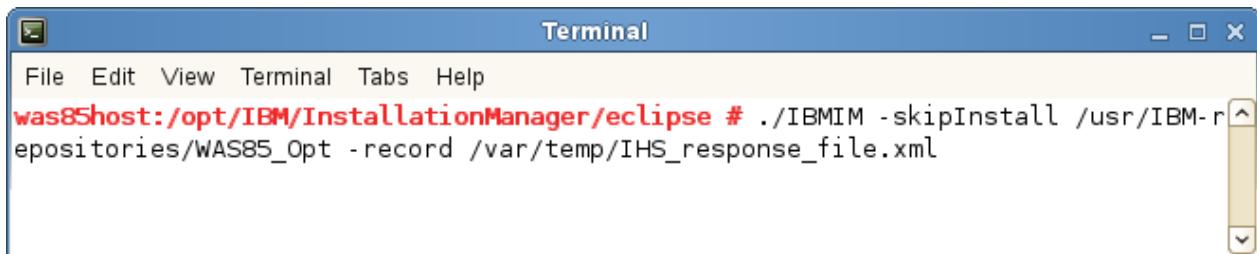
```
./IBMMIM -skipInstall user_home/var/temp/imRegistry -record user_home/var/temp/install_response_file.xml
```

**Tip:** When you record a new response file, you can specify the `-skipInstall` parameter. Using this parameter has the following benefits:

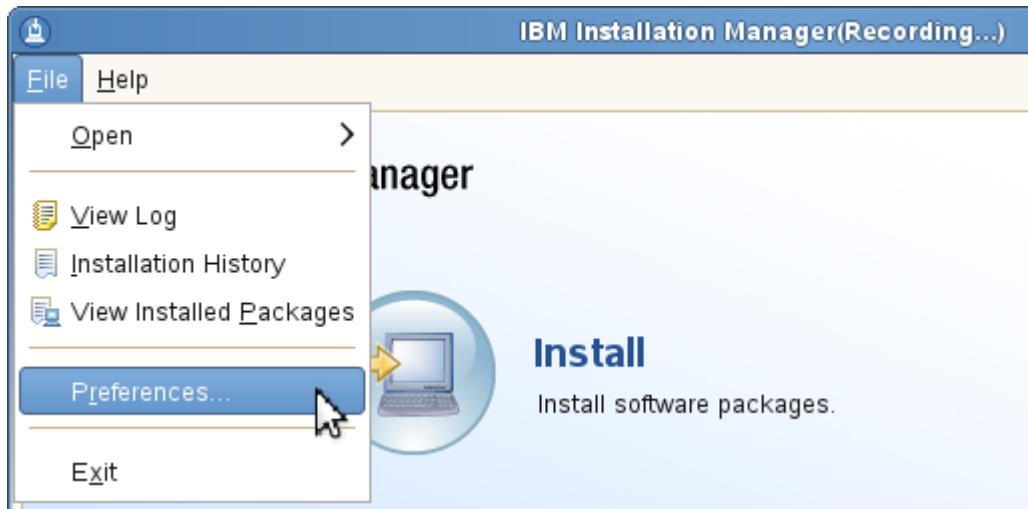
- No files are actually **installed**, and this speeds up the recording.
- If you use a temporary data location with the `-skipInstall` parameter, **Installation Manager** writes the **installation registry** to the specified data location while recording. When you start **Installation Manager** again without the `-skipInstall` parameter, you then can use your response file to **install** against the real **installation registry**.

- \_\_ a. Open a terminal window and go to  
`/opt/IBM/InstallationManager/eclipse`
- \_\_ b. Start the IBM Installation Manager in record mode by running the following command:

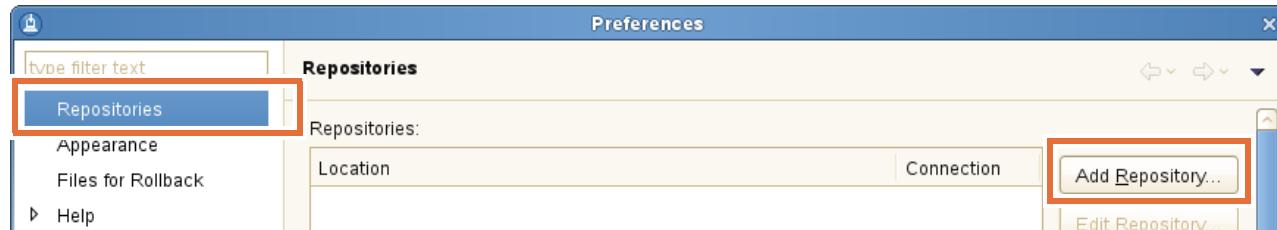
```
./IBMIM -skipInstall /usr/IBM-repositories/WAS85_Opt  
-record /var/temp/IHS_response_file.xml
```



- \_\_ c. Wait for the IBM Installation Manager to start, and select **File > Preference**.

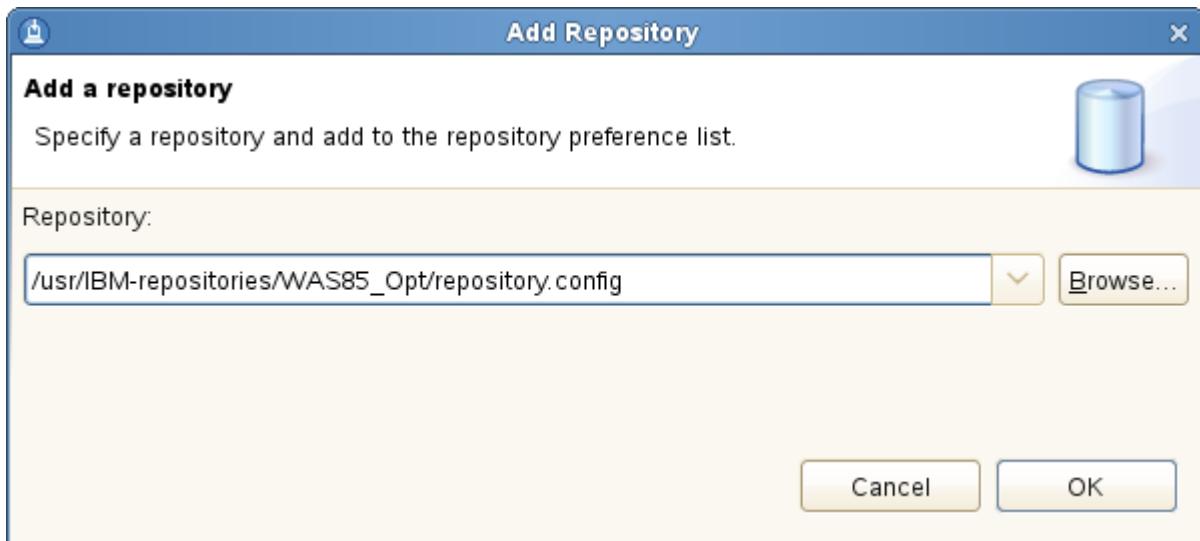


- \_\_ d. Select Repositories, and click **Add Repository**.

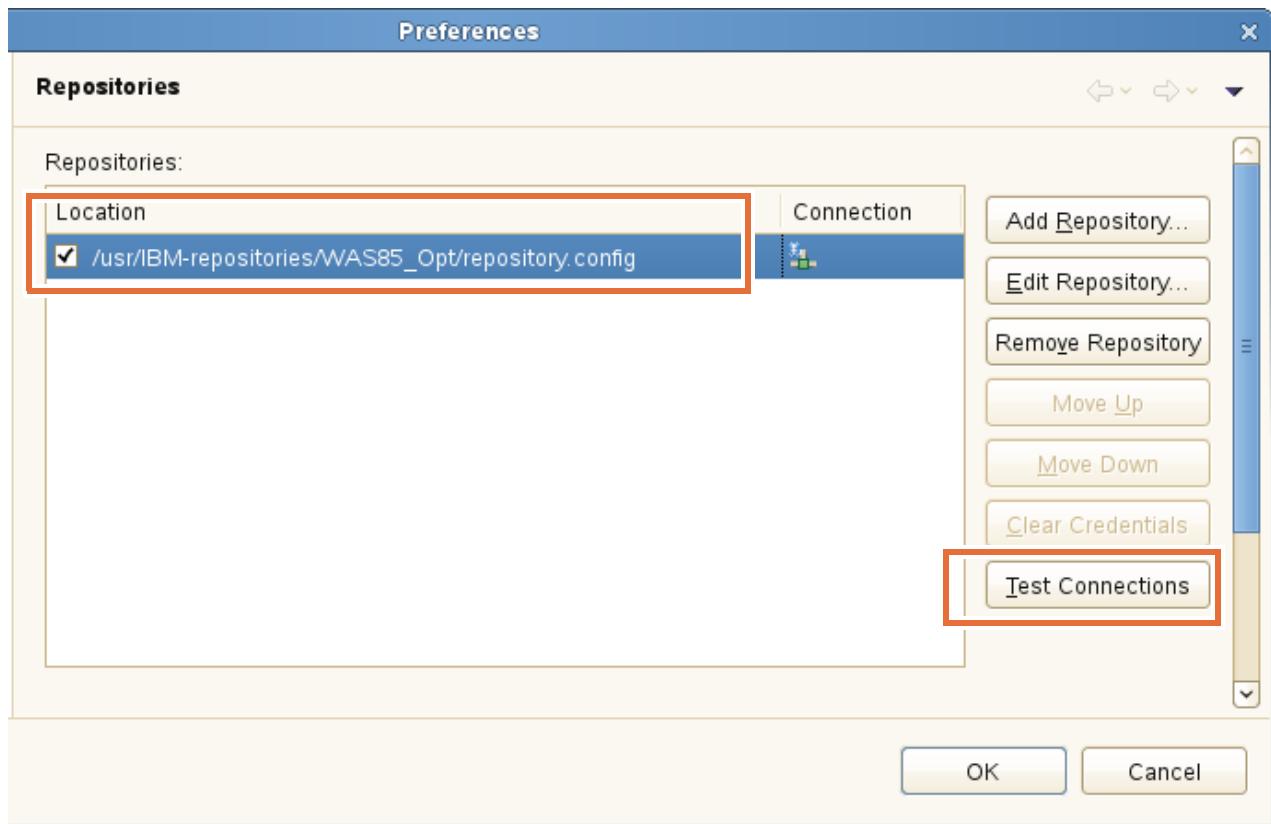


- \_\_ e. Browse to **/usr/IBM-repositories/WAS85\_Opt**

\_\_\_ f. Select **repository.config**, and click **OK**.



\_\_\_ g. Back on the Add a repository pane, click **OK**.

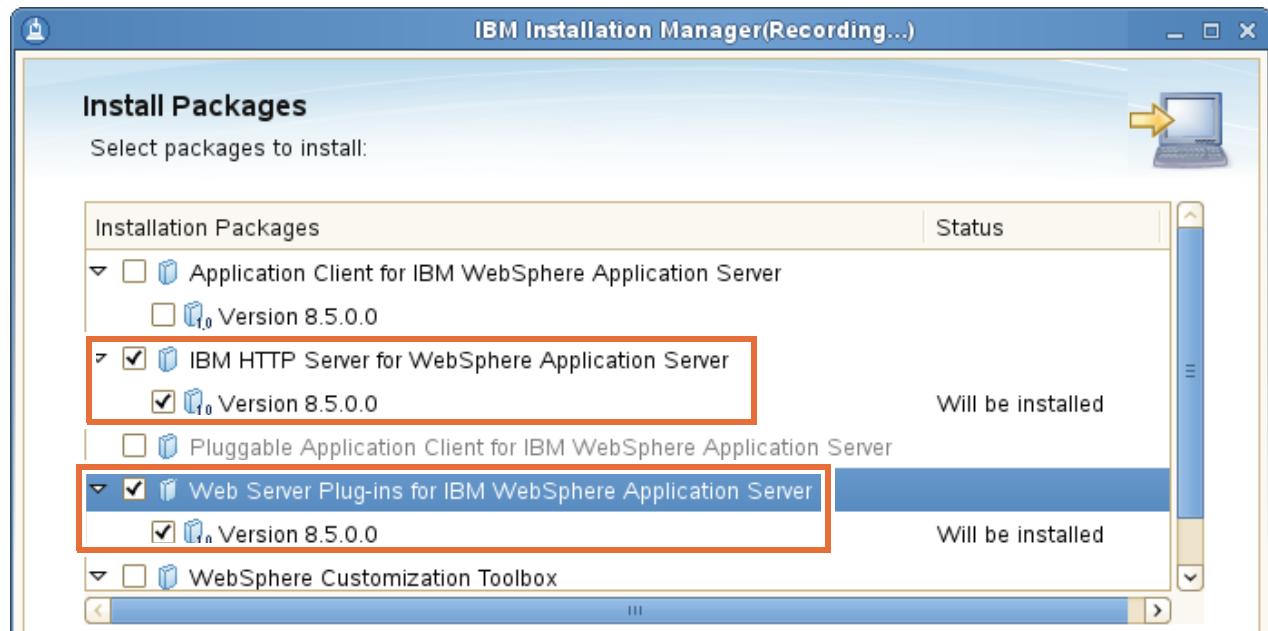


\_\_\_ h. Verify that the location for the repository is added to the list. If the repository is remote, it is a good idea to click **Test Connections** to also verify that IBM Installation Manager can connect to the repository. Click **OK**.

- \_\_\_ i. Back on the IBM Installation Manager welcome page, click **Install**. Notice that the IBM Installation Manager is running in record mode.

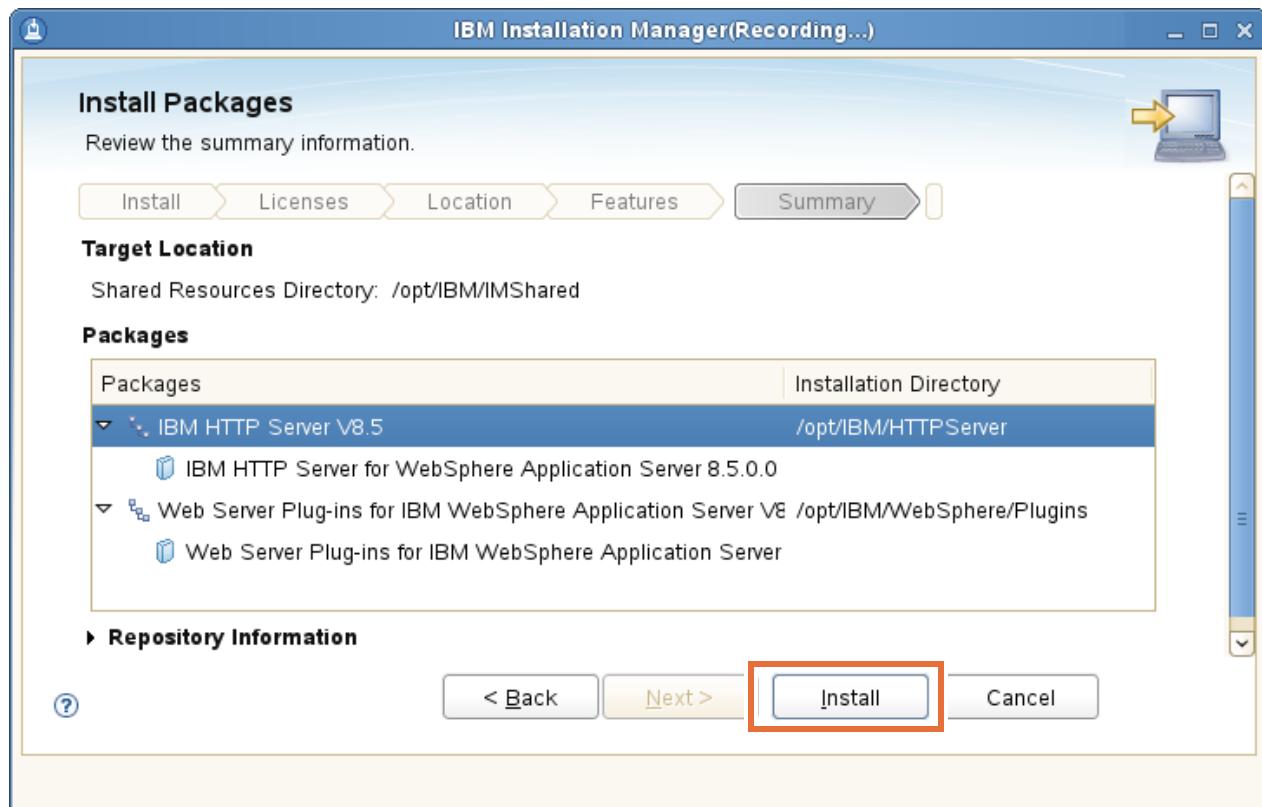


- \_\_\_ j. Select both **IBM HTTP Server for WebSphere Application Server** and **Web Server Plug-ins for IBM WebSphere Application Server**.



- \_\_\_ k. Click **Next**.
- \_\_\_ l. Accept the License agreements, and click **Next**. You are going to accept all of the default settings on the next several screens.
- \_\_\_ m. Keep the default shared resource directory `/opt/IBM/IMShared`, click **Next**.
- \_\_\_ n. Keep the default installation directory `/opt/IBM/HTTPServer`, and click **Next**.
- \_\_\_ o. Click **Next** on Features.
- \_\_\_ p. Keep HTTP port **80**, and click **Next**.

- \_\_ q. On the Install Packages pane, click **Install**.



- \_\_ r. The “Installation” happens quickly since the products are not installed because of the `-skipInstall` command-line argument. Click **Finish**.
- \_\_ s. Exit IBM Installation Manager by clicking **File > Exit**.
- \_\_ 2. Examine the response file that the IBM Installation Manager generates.
- \_\_ a. From a terminal window, go to `/var/temp`

- \_\_ b. Use a text editor (such as gedit) to open the **IHS\_response\_file.xml** file.

```

<?xml version="1.0" encoding="UTF-8"?>
<!--The "acceptLicense" attribute has been deprecated. Use "-acceptLicense" command line option to accept license agreements.-->

<server>
<repository location='/usr/IBM-repositories/WAS85_Opt' />
</server>
<profile id='IBM HTTP Server V8.5' installLocation='/opt/IBM/HTTPServer'>
<data key='eclipseLocation' value='/opt/IBM/HTTPServer' />
<data key='user.import.profile' value='false' />
<data key='cic.selector.os' value='linux' />
<data key='cic.selector.ws' value='gtk' />
<data key='cic.selector.arch' value='x86' />
<data key='user.ihs.http.server.service.name' value='none' />
<data key='user.ihs.httpPort' value='80' />
<data key='user.ihs.installHttpService' value='false' />
<data key='cic.selector.nl' value='en' />
</profile>

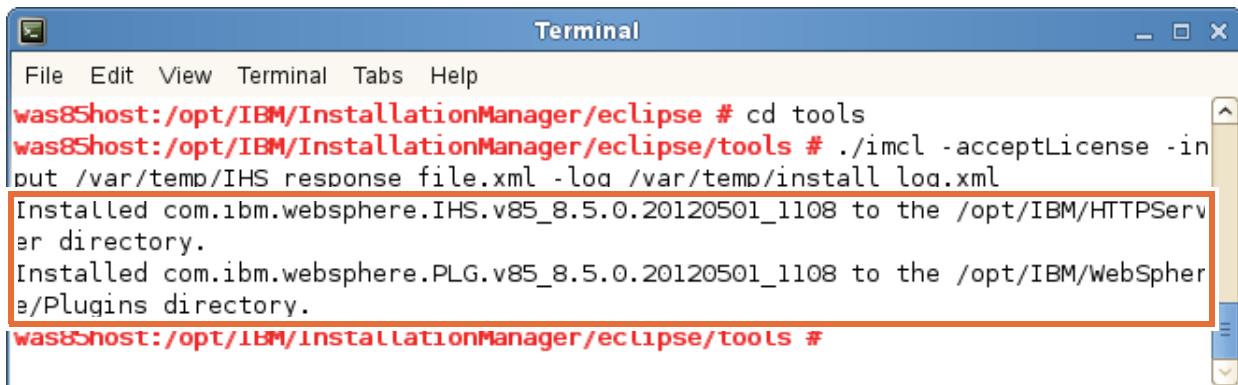
<install modify='false'>
<offering id='com.ibm.websphere.IHS.v85' version='8.5.0.20120501_1108' profile='IBM HTTP Server V8.5' features='core.feature,arch.32bit' installFixes='none' />
<offering id='com.ibm.websphere.PLG.v85' version='8.5.0.20120501_1108' profile='Web Server Plug-ins for IBM WebSphere Application Server V8.5' features='core.feature.com.ibm.iec.6_32bit' installFixes='none' />
</install>
<profile id='Web Server Plug-ins for IBM WebSphere Application Server V8.5' installLocation='/opt/IBM/WebSphere/Plugins'>
<data key='eclipseLocation' value='/opt/IBM/WebSphere/Plugins' />
<data key='user.import.profile' value='false' />
<data key='cic.selector.os' value='linux' />
<data key='cic.selector.ws' value='gtk' />
<data key='cic.selector.arch' value='x86' />
<data key='cic.selector.nl' value='en' />
</profile>

```

The top sections of the response file show the installation options for the IBM HTTP Server and the Web Server Plug-in.

- \_\_ c. Close the **IHS\_response\_file.xml** file.
- \_\_ 3. Run the Installation Manager command line (**imcl**) command to install IBM HTTP Server for WebSphere Application Server and the Web Server Plug-in.
- \_\_ a. From a terminal window, go to  
**/opt/IBM/InstallationManager/eclipse/tools**
- \_\_ b. Run the following command:

```
./imcl -acceptLicense -input /var/temp/IHS_response_file.xml  
-log /var/temp/install_log.xml
```



A screenshot of a terminal window titled "Terminal". The window shows a command-line session. The user has navigated to the "/opt/IBM/InstallationManager/eclipse/tools" directory and run the command "./imcl -acceptLicense -input /var/temp/IHS\_response\_file.xml -log /var/temp/install\_log.xml". The output of the command is displayed in red text, indicating successful installations of the com.ibm.websphere.IHS.v85\_8.5.0.20120501\_1108 plugin to the "/opt/IBM/HTTPServer" directory and the com.ibm.websphere.PLG.v85\_8.5.0.20120501\_1108 plugin to the "/opt/IBM/WebSphere/Plugins" directory.

```
File Edit View Terminal Tabs Help  
was85host:/opt/IBM/InstallationManager/eclipse # cd tools  
was85host:/opt/IBM/InstallationManager/eclipse/tools # ./imcl -acceptLicense -in  
put /var/temp/IHS response file.xml -log /var/temp/install log.xml  
Installed com.ibm.websphere.IHS.v85_8.5.0.20120501_1108 to the /opt/IBM/HTTPServ  
er directory.  
Installed com.ibm.websphere.PLG.v85_8.5.0.20120501_1108 to the /opt/IBM/WebSpher  
e/Plugins directory.  
was85host:/opt/IBM/InstallationManager/eclipse/tools #
```

- \_\_\_ c. After a minute or two, verify that you see the messages which state that IHS installed to the /opt/IBM/HTTPServer directory and PLG (the plug-in) is installed to the /opt/IBM/WebSphere/Plugins directory.

## Section 2: Run a script to configure the Web Server Plug-in.



### Information

To find out how to configure the Web Server Plug-in, search the WebSphere Application Server Information Center for: **Configuring a web server plug-in using the pct tool**.

[Implementing a web server plug-in](#) > [Installing and configuring web server plug-ins](#)

#### Configuring a web server plug-in using the pct tool

The WCT command invokes a command-line tool that is specified by the `-tool` parameter. You can use the WCT command and specify the `pct` tool to configure a web server to use an application server as a hosting server.

#### Procedure

Configure a web server to use an application server as a hosting server.

#### Location of the WCT command

The product includes the following script that sets up the environment and invokes the WCT command.

- Windows `WCT_install_root\WCT\wctcmd.bat`
- Linux `WCT_install_root/WCT/wctcmd.sh`

#### Syntax of the WCT command when invoking the pct tool

##### Windows

```
wctcmd.bat
  -tool pct
  -defLocPathname definition_location_pathname
  -defLocName definition_location_name
  -createDefinition definition_name
  -deleteDefinition definition_name
  -response response_file
  -removeDefinitionLocation definition_location_name
```

##### Linux

```
./wctcmd.sh
  -tool pct
  -defLocPathname definition_location_pathname
  -defLocName definition_location_name
  -createDefinition definition_name
  -deleteDefinition definition_name
  -response response_file
  -removeDefinitionLocation definition_location_name
```

You can run the `wctcmd` command to silently configure a web server plug-in by providing a response file.

**Examples**

**Using the pct tool to configure an IHS Web Server to use an application server as a hosting server:**

- **Windows** wctcmd.bat -**tool** pct -**defLocPathname** C:\data\IBM\WebSphere\Plugins -**defLocName** someDefinitionLocationName -**createDefinition** someDefinitionName -**response** C:\IBM\WebSphere\tools\WCT\responsefile.txt
- **Linux** ./wctcmd.sh -**tool** pct -**defLocPathname** /data/IBM/WebSphere/Plugins -**defLocName** someDefinitionLocationName -**createDefinition** someDefinitionName -**response** /var/IBM/WebSphere/tools/WCT/responsefile.txt

**Parameters of the WCT command when starting the pct tool****-tool** pct

Specifies the name of the tool to start as it is registered with the WCT command. This parameter is required.

**-defLocPathname** definition\_location\_pathname

Specifies the absolute path name of the existing plug-in runtime root location. This parameter is required when you create or delete a definition.

**-defLocName** definition\_location\_name

Specifies the logical name of the plug-in runtime root location. This parameter is required when you create or delete a definition.

**-createDefinition** definition\_name

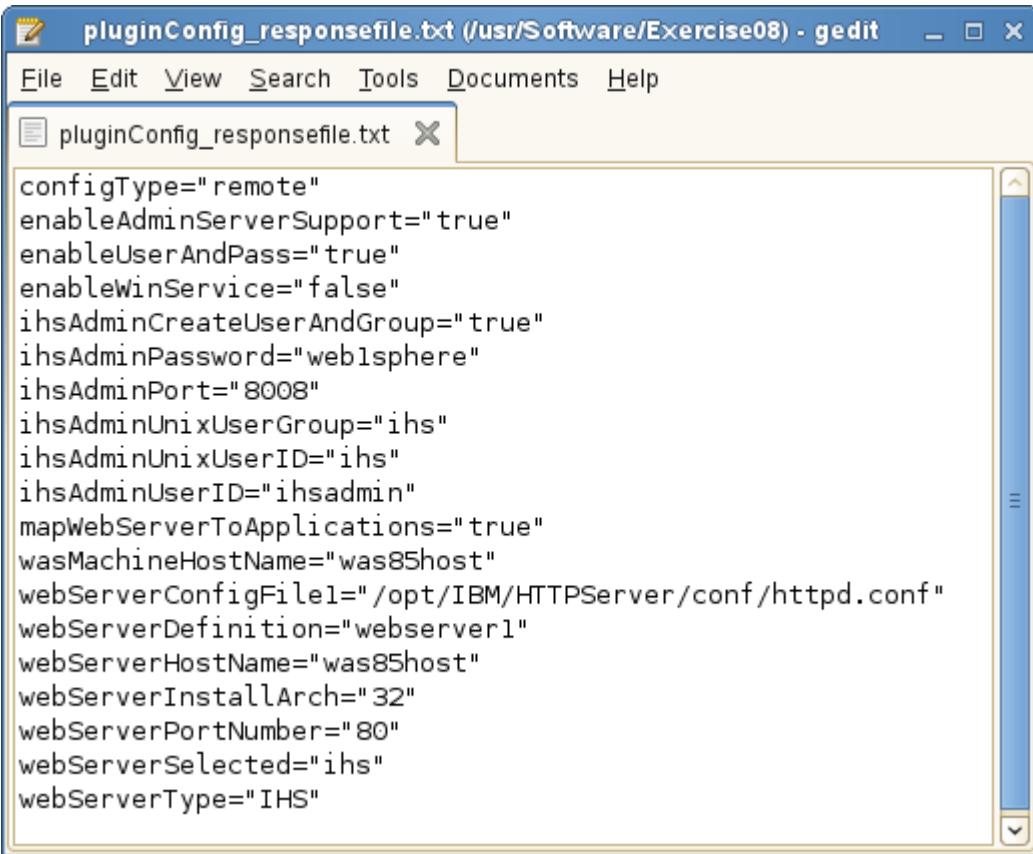
Specifies the unique web server definition name to be used with the configuration. This parameter is used when you create a definition.

**-response** response\_file

Specifies the response file that contains tool arguments. This parameter is required when you create a definition.

- 1. A response file for the web server plug-in is provided for you. Explore the response file.
  - a. Go to **/usr/Software/Scripts/Exercise08**

- \_\_ b. Open `pluginConfig_responsefile.txt` by using a text editor such as gedit.



The screenshot shows a Gedit text editor window with the title bar "pluginConfig\_responsefile.txt (/usr/Software/Exercise08) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. A tab bar below the title bar shows "pluginConfig\_responsefile.txt" and a close button. The main text area contains the following configuration options:

```
configType="remote"
enableAdminServerSupport="true"
enableUserAndPass="true"
enableWinService="false"
ihsAdminCreateUserAndGroup="true"
ihsAdminPassword="websphere"
ihsAdminPort="8008"
ihsAdminUnixUserGroup="ihs"
ihsAdminUnixUserID="ihs"
ihsAdminUserID="ihsadmin"
mapWebServerToApplications="true"
wasMachineHostName="was85host"
webServerConfigFile1="/opt/IBM/HTTPServer/conf/httpd.conf"
webServerDefinition="webserver1"
webServerHostName="was85host"
webServerInstallArch="32"
webServerPortNumber="80"
webServerSelected="ihs"
webServerType="IHS"
```

- \_\_ c. Examine the required options and their values  
\_\_ d. Close the response file.

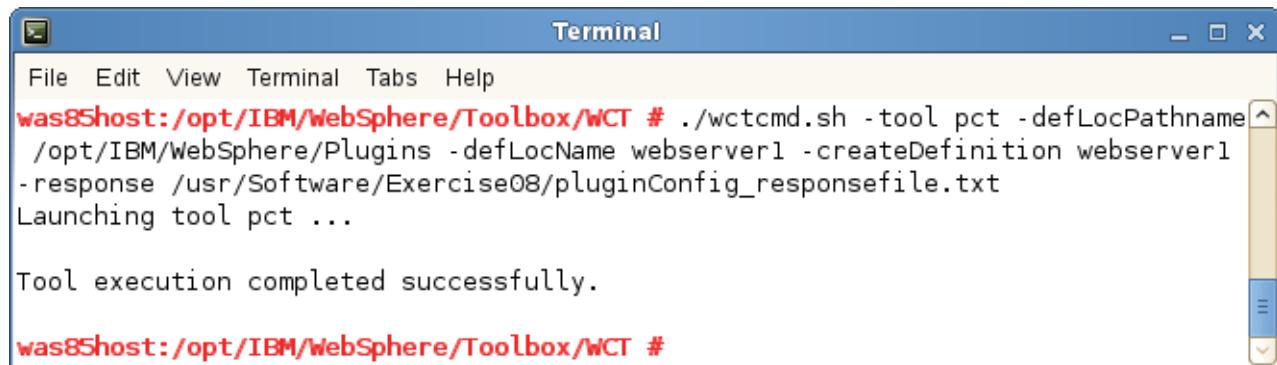


### Information

#### Response file options

A description of the required options for this response file is provided in the template at `/opt/IBM/WebSphere/Toolbox/WCT/pct_response.txt`. This template is created when the WebSphere Customization Toolbox is installed.

2. Run the `wctcmd` command.
- From a terminal window, go to `/opt/IBM/WebSphere/Toolbox/WCT`
  - Run the following command all on one line, or copy it from the `/usr/Solutions/Exercise08/cmd.txt` file.
- ```
./wctcmd.sh -tool pct -defLocPathname /opt/IBM/WebSphere/Plugins
             -defLocName webserver1 -createDefinition webserver1 -response
             /usr/Software/Scripts/Exercise08/pluginConfig_responsefile.txt
```

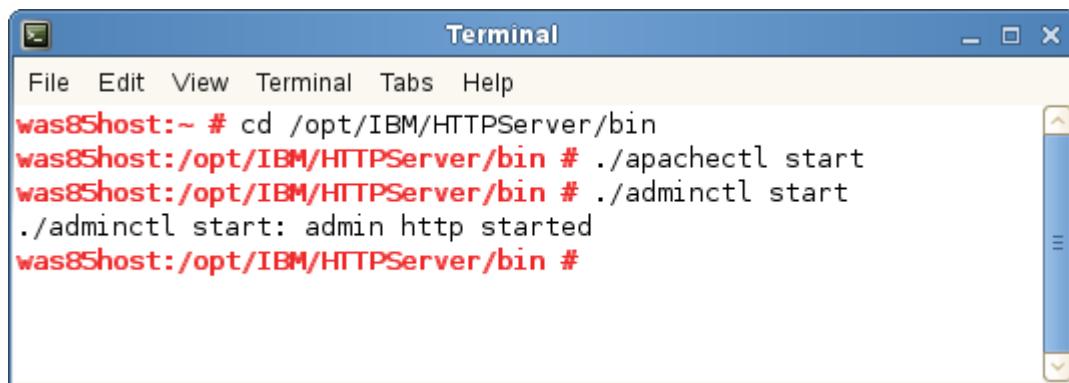


```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/Toolbox/WCT # ./wctcmd.sh -tool pct -defLocPathname
  /opt/IBM/WebSphere/Plugins -defLocName webserver1 -createDefinition webserver1
  -response /usr/Software/Exercise08/pluginConfig_responsefile.txt
Launching tool pct ...

Tool execution completed successfully.

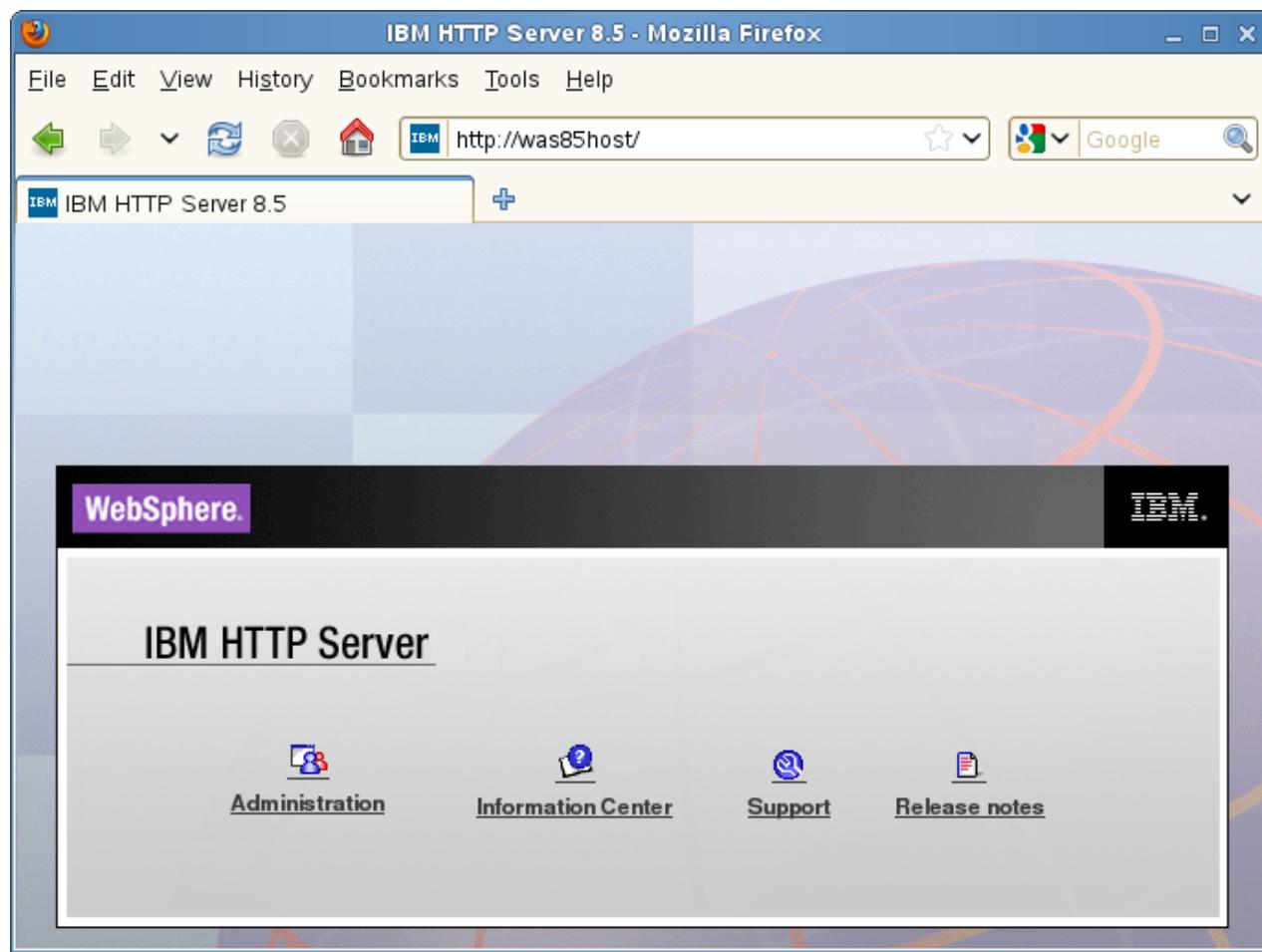
was85host:/opt/IBM/WebSphere/Toolbox/WCT #
```

3. Start the IBM HTTP Server and the IBM HTTP Server administration server.
- From a terminal window, go to `/opt/IBM/HTTPServer/bin`
  - Start the IBM HTTP Server with the following command: `./apachectl start`
  - Start the IBM HTTP Server administration server with: `./adminctl start`



```
File Edit View Terminal Tabs Help
was85host:~ # cd /opt/IBM/HTTPServer/bin
was85host:/opt/IBM/HTTPServer/bin # ./apachectl start
was85host:/opt/IBM/HTTPServer/bin # ./adminctl start
./adminctl start: admin http started
was85host:/opt/IBM/HTTPServer/bin #
```

- \_\_\_ 4. Verify that the IBM HTTP Server is working.
  - \_\_\_ a. Open a web browser and enter the web address: **http://was85host**



The IBM HTTP Server 8.5 administration home page is displayed.

Success! You completed the silent installation of the IBM HTTP Server and the web server plug-in.

- \_\_\_ b. Close the web browser.

## **Section 3: Creating an unmanaged node and web server definition by using scripting**

A useful feature of WebSphere Application Server is the ability to control and manage an IBM HTTP Server installation without the need of a node agent. This feature is implemented by creating an *unmanaged* node in the configuration for it. In this case, WebSphere Application Server uses the IBM HTTP Server administration service to manage the web server. Additionally, by defining the web server to the configuration, you can generate and automatically propagate the plug-in file for the web server.

In this step, you develop a parameterized Jython function that creates an unmanaged node for a web server, and defines the web server to the configuration. You then start it to create the *ihsnode* and *webserver1* web server resources.

### **Requirements specification**

Your specific requirements are as follows:

1. Create a Jython function named `createWebServer` to create an unmanaged node and a web server definition given the following as parameters:
  - `nodeName`: Name that represents the node in the configuration
  - `hostName`: Host name of system that is associated with node
  - `nodeOperatingSystem`: Name of operating system in use on the system that is associated with node. Valid entries include the following systems: `os400`, `aix`, `hpxx`, `linux`, `solaris`, `windows`, and `os390`
  - `webServerName`: Name of the web server
  - `templateName`: Name of the template that you want to use. Templates values include: `IHS`, `iPlanet`, `IIS`, `DOMINO`, `APACHE`.
  - `adminPort`: Port number of the IBM HTTP Server administration server
  - `ihsAdminUserID`: User ID used to authenticate to the IBM HTTP Server administration service
  - `ihsAdminPassword`: Password that is used to authenticate to the IBM HTTP Server administration service
  - `pluginInstallRoot`: Specifies the installation path for the plug-in

The first three parameters are used to create the unmanaged node, while the first and last five are used to create the web server definition.

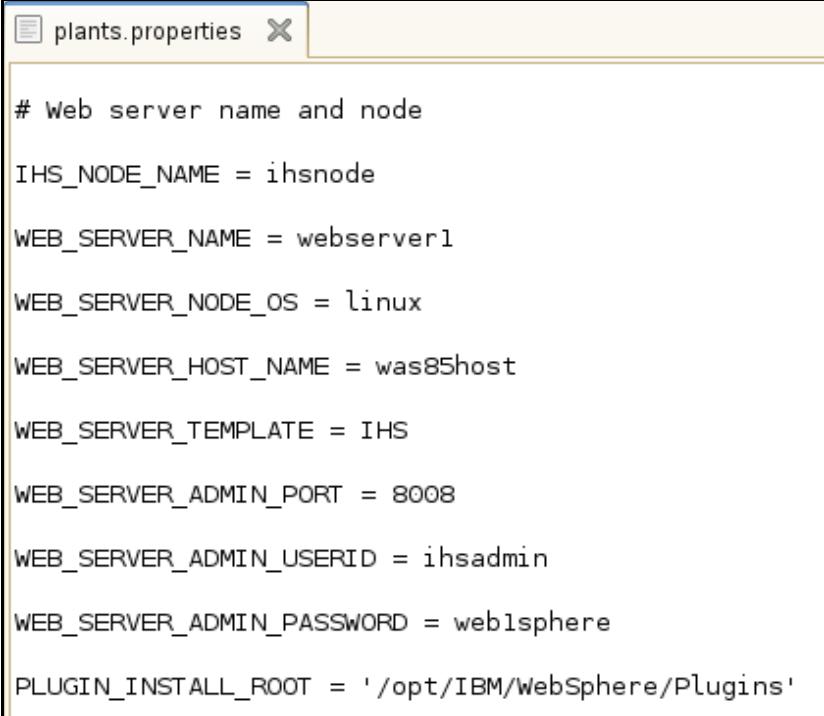
2. Start the function to create the *ihsnode* and *webserver1* definitions with the following options:

**Table 6: Parameters for ihsnode and webserver1 definitions**

|   | <b>Option name</b>  | <b>Option value</b>                                                                      |
|---|---------------------|------------------------------------------------------------------------------------------|
| 1 | nodeName            | Retrieve the value of property named "IHS_NODE_NAME" from plants.properties.             |
| 2 | hostName            | Retrieve the value of property named "WEB_SERVER_HOST_NAME" from plants.properties.      |
| 3 | nodeOperatingSystem | Retrieve the value of property named "WEB_SERVER_NODE_OS" from plants.properties.        |
|   | webServerName       | Retrieve the value of property named "WEB_SERVER_NAME" from plants.properties.           |
| 4 | templateName        | Retrieve the value of property named "WEB_SERVER_TEMPLATE" from plants.properties.       |
| 5 | adminPort           | Retrieve the value of property named "WEB_SERVER_ADMIN_PORT" from plants.properties.     |
| 6 | ihsAdminUserID      | Retrieve the value of property named "WEB_SERVER_ADMIN_USERID" from plants.properties.   |
| 7 | ihsAdminPassword    | Retrieve the value of property named "WEB_SERVER_ADMIN_PASSWORD" from plants.properties. |
| 8 | pluginInstallRoot   | Retrieve the value of property named "PLUGIN_INSTALL_ROOT" from plants.properties.       |

- \_\_\_ 1. All of the parameters for creating the ihsnode and webserver1 are in the plants.properties file. Open the properties file to examine the values for each parameter.
  - \_\_\_ a. From a terminal window, go to **/usr/Software/Scripts/Properties**.

- \_\_\_ b. Use a text editor to open the `plants.properties` file.



```
# Web server name and node
IHS_NODE_NAME = ihsnode
WEB_SERVER_NAME = webserver1
WEB_SERVER_NODE_OS = linux
WEB_SERVER_HOST_NAME = was85host
WEB_SERVER_TEMPLATE = IHS
WEB_SERVER_ADMIN_PORT = 8008
WEB_SERVER_ADMIN_USERID = ihsadmin
WEB_SERVER_ADMIN_PASSWORD = web1sphere
PLUGIN_INSTALL_ROOT = '/opt/IBM/WebSphere/Plugins'
```

- \_\_\_ c. Verify that all of the required parameters are defined and note their values.  
 \_\_\_ d. Close the properties file.

## Creating the `createWebServer` function

- \_\_\_ 1. Start the IBM Assembly and Deploy Tools and open a new workspace.
  - \_\_\_ a. On the desktop, double-click the **IBM Assembly and Deploy Tools** icon. If there is no IADT icon, open a terminal window, go to `/opt/IBM/SDP`, and enter the command `./eclipse &`
  - \_\_\_ b. In the Workspace Launcher window, select `/usr/LabWork/DistributedWS` in the Workspace field.
  - \_\_\_ c. Click **OK**. The IADT opens and displays the Java EE perspective by default.
- \_\_\_ 2. Import the skeleton function and mainline scripts that are designed to help you in your script development.
  - \_\_\_ a. In the Enterprise Explorer view, expand the **PlantsScriptingProject** folder, right-click **Scripts**, and select **Import**.
  - \_\_\_ b. In the *Import select* dialog, expand **General** and select **File system**.
  - \_\_\_ c. Click **Next**.
  - \_\_\_ d. In the *File system* dialog, click **Browse** next to the From directory field.

- \_\_\_ e. In the *Import from directory* dialog, go to **/usr/Software/Scripts**, select **Exercise08**, and click **OK**.
  - \_\_\_ f. Back in the *File system* dialog, select only the following files from the folder:
    - ex08\_createWebServer.py
    - ut08\_createWebServer.py
  - \_\_\_ g. Click **Finish**.
  - \_\_\_ h. In the Enterprise Explorer view, expand **PlantsScriptingProject > Scripts**. Verify that the skeleton function and mainline scripts for this exercise are in the **Scripts** folder.
- \_\_\_ 3. Open the `ex08_createWebServer.py` function skeleton script in a Jython editor view.
- \_\_\_ a. Enable the line numbers by right-click anywhere in the editor view and select **Preferences**.
  - \_\_\_ b. Check the box for **Show line numbers**, and click **OK**.

This file provides a function definition statement and general instructions in the form of comments to help complete the code. It also includes basic print statements that display execution progress messages.

```

 ex08_createWebServer.py ☒
1 # ex08_createWebServer.py
2 def createWebServer(nodeName,
3                     hostName,
4                     nodeOperatingSystem,
5                     webServerName,
6                     templateName,
7                     adminPort,
8                     ihsAdminUserID,
9                     ihsAdminPassword
10                    pluginInstallRoot):
11
12     print "Creating", nodeName, "unmanaged node"
13     #Build parameter list for AdminTask createUnmanagedNode command.
14     unmanagedNodeParms = []
15
16     # Call AdminTask to create the unmanaged node.
17     AdminTask.createUnmanagedNode(unmanagedNodeParms)
18
19     print nodeName, "unmanaged node created"
20     print "Creating Web server definition for", webServerName, "on", nodeName
21
22     #Build options list for -remoteServerConfig step.
23     remoteServerOptions = []
24
25     #Build options list for -serverConfig step.
26     serverConfig = []
27
28     # Call AdminTask to create the Web server definition on nodeName.
29     AdminTask.
30
31     print "Web server definition for", webServerName, "on", nodeName, "created"
32     # Save the configuration changes.
33     #
34     #AdminConfig.save()
35

```

4. Complete the code as instructed in the comments in lines 13 and 28. The AdminTask command to use was already identified for you: *createUnmanagedNode*. Use the Information Center and other available resources to determine the command parameters and correct invocation syntax.



### Hint

The *createUnmanagedNode* AdminTask command is part of the *UnmanagedNodeCommands* command group. Search the Information Center, and use the description and examples that are provided to figure out the parameters that it requires. You can also get more usage details by using one or more of the following methods:

- *AdminTask.help(aCommandName)*
- *AdminTask.help(aCommandName, aStepName)*
- *AdminTask.aCommandName("-interactive")* (runs the command in interactive mode)

## UnmanagedNodeCommands command group for the AdminTask object using wsadmin scripting

You can use the Jython or Jacl scripting languages to manage servers with the wsadmin tool. The commands and parameters in the **UnmanagedNodeCommands** group can be used to create and query for managed and unmanaged nodes. An unmanaged node is a node that does not have a node agent or a deployment manager.

The **UnmanagedNodeCommands** command group for the AdminTask object includes the following commands:

- [createUnmanagedNode](#)
- [listManagedNodes](#)
- [listUnmanagedNodes](#)
- [removeUnmanagedNode](#)

### **createUnmanagedNode**

Use the **createUnmanagedNode** command to create a new unmanaged node in the configuration. An unmanaged node is a node that does not have a node agent or a deployment manager. Unmanaged nodes can contain web servers, such as IBM® HTTP Server.

#### Target object

None

#### Parameters and return values

**-nodeName**

The name that will represent the node in the configuration repository. (String, required)

**-hostName**

The host name of the system associated with this node. (String, required)

**-nodeOperatingSystem**

The operating system in use on the system associated with this node. Valid entries include the following:

`os400, aix, hpx, linux, solaris, windows, and os390.` (String required)

#### Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createUnmanagedNode {-nodeName myNode -hostName myHost
                               -nodeOperatingSystem linux}
```

- Using Jython string:

```
AdminTask.createUnmanagedNode(['-nodeName jjNode -hostName jjHost
                               -nodeOperatingSystem linux'])
```

- Using Jython list:

```
AdminTask.createUnmanagedNode(['-nodeName', 'jjNode', '-hostName', 'jjHost',
                               '-nodeOperatingSystem', 'linux'])
```

5. Complete the code as instructed in the comments in lines 22 and 25. You now must identify the appropriate AdminTask command to use, understand its target, arguments, and steps, and compose its invocation syntax.



### Hint

Search the Information Center for commands that belong to the *ServerManagement* command group.

After you identify the command to use, use the description and examples that are provided to figure out the target, arguments, and steps that it requires. You can also get more usage details by using one or more of the following methods:

- `AdminTask.help(aCommandName)`
- `AdminTask.help(aCommandName, aStepName)`
- `AdminTask.aCommandName("-interactive")` (runs the command in interactive mode)

[Network Deployment \(Distributed operating systems\), Version 8.5 > Reference > Commands \(wsadmin scripting\)](#)

### ServerManagement command group for the AdminTask object

You can use the Jython or Jacl scripting languages to manage servers with the wsadmin tool. The **commands** and parameters in the **ServerManagement** group can be used to create and manage application server, web server, proxy server, generic server and Java virtual machine (JVM) configurations.

The **ServerManagement command group for the AdminTask object** includes the following **commands**:

- [createApplicationServer](#)
- [createApplicationServerTemplate](#)
- [createGenericServer](#)
- [createGenericServerTemplate](#)
- [createGenericServerTemplate](#)
- [createProxyServer](#)
- [createProxyServerTemplate](#)
- [createServerType \(Deprecated\)](#)
- [createWebServer](#)
- [createWebServerTemplate](#)
- [deleteServer](#)

Batch mode example usage:

- Using Jython list:

```
print AdminTask.createWebServer(myNode, ['-name', wsname,
    '-serverConfig',[['80','/opt/path/to/ihc', '/opt/path/to/plugin',
    '/opt/path/to/plugin.xml', 'windows service','/opt/path/to/error.log',
    '/opt/path/to/access.log','HTTP']],'-remoteServerConfig', [['8008','user',
    'password', 'HTTP']]])
```

- Using Jython string:

```
AdminTask.createWebServer('myNode','-name wsname -serverConfig [80
    /opt/path/to/ihc /opt/path/to/plugin /opt/path/to/plugin.xml "windows service"
    /opt/path/to/error.log /opt/path/to/access.log HTTP]
    -remoteServerConfig [8008 user password HTTP]')
```

Your completed *createWebServer* function should look as follows:

```
ex08_createWebServer.py
```

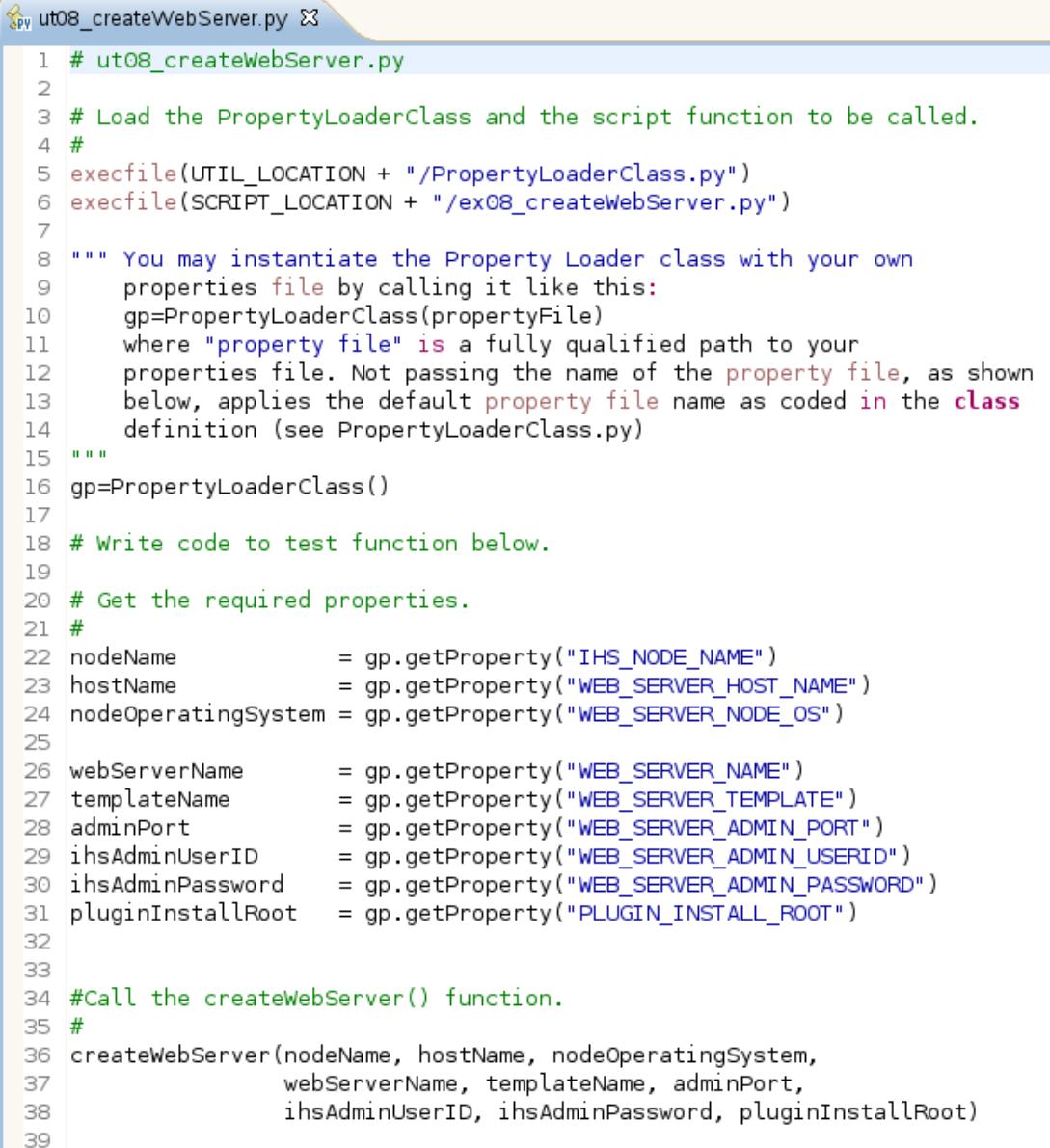
```
1 # ex08_createWebServer.py
2 def createWebServer(nodeName,
3                     hostName,
4                     nodeOperatingSystem,
5                     webServerName,
6                     templateName,
7                     adminPort,
8                     ihsAdminUserID,
9                     ihsAdminPassword,
10                    pluginInstallRoot):
11
12     print "Creating", nodeName, "unmanaged node"
13     #Build parameter list for AdminTask createUnmanagedNode command.
14     unmanagedNodeParms = ["-nodeName", nodeName, "-hostName", hostName,
15                           "-nodeOperatingSystem", nodeOperatingSystem]
16
17     # Call AdminTask to create the unmanaged node.
18     AdminTask.createUnmanagedNode(unmanagedNodeParms)
19     print nodeName, "unmanaged node created"
20     print "Creating Web server definition for", webServerName, "on", nodeName
21
22     #Build options list for -remoteServerConfig step.
23     remoteServerOptions = [adminPort, ihsAdminUserID, ihsAdminPassword]
24
25     #Build options list for -serverConfig step.
26     serverConfig = ["-pluginInstallRoot", pluginInstallRoot]
27
28     # Call AdminTask to create the Web server definition on nodeName.
29     AdminTask.createWebServer(nodeName, ["-name", webServerName, "-templateName", templateName,
30                                     "-remoteServerConfig", [remoteServerOptions],"-serverConfig", serverConfig])
31
32     print "Web server definition for", webServerName, "on", nodeName, "created"
33     # Save the configuration changes.
34     #
35     #AdminConfig.save()
```

## 6. Save your changes.

## Starting the `createWebServer` function

- \_\_\_ 1. Use the `ut08_createWebServer.py` script to start the `createWebServer` function that you created. Modify it to:
  - \_\_\_ a. Retrieve any properties that are required to create the `ihsnode` and `webserver1` definitions as listed in the options table, by using the `gp.getProperty(propertyName)` method.
  - \_\_\_ b. Start the `createWebServer` function to create the `ihsnode` and `webserver1` definitions with the parameter values specified in the options table.

Your `ut08_createWebServer.py` script should look as follows:



```

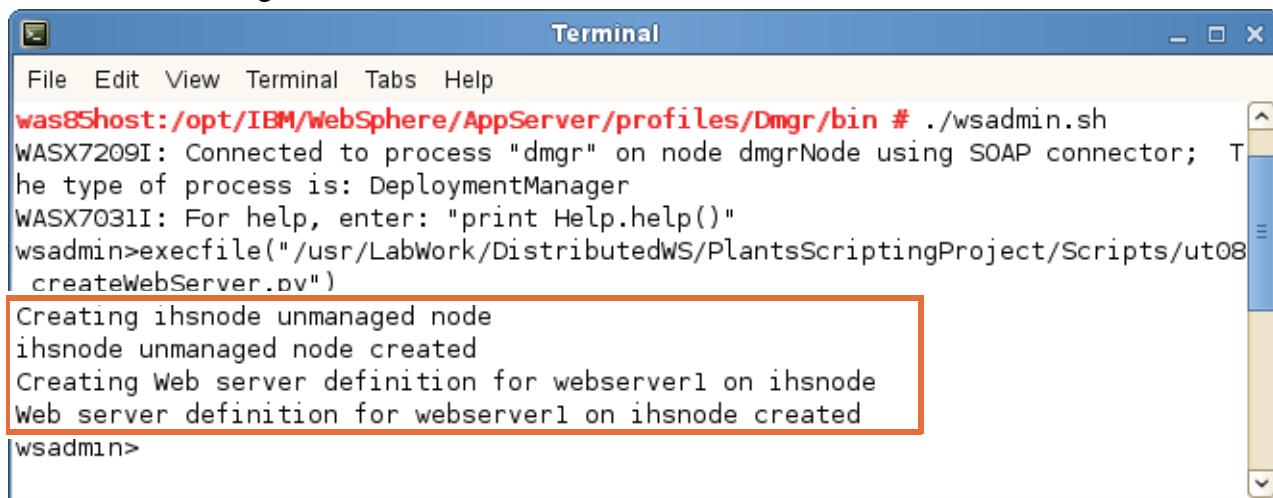
1 # ut08_createWebServer.py
2
3 # Load the PropertyLoaderClass and the script function to be called.
4 #
5 execfile(UTIL_LOCATION + "/PropertyLoaderClass.py")
6 execfile(SCRIPT_LOCATION + "/ex08_createWebServer.py")
7
8 """ You may instantiate the Property Loader class with your own
9 properties file by calling it like this:
10 gp=PropertyLoaderClass(propertyFile)
11 where "property file" is a fully qualified path to your
12 properties file. Not passing the name of the property file, as shown
13 below, applies the default property file name as coded in the class
14 definition (see PropertyLoaderClass.py)
15 """
16 gp=PropertyLoaderClass()
17
18 # Write code to test function below.
19
20 # Get the required properties.
21 #
22 nodeName          = gp.getProperty("IHS_NODE_NAME")
23 hostName          = gp.getProperty("WEB_SERVER_HOST_NAME")
24 nodeOperatingSystem = gp.getProperty("WEB_SERVER_NODE_OS")
25
26 webServerName     = gp.getProperty("WEB_SERVER_NAME")
27 templateName      = gp.getProperty("WEB_SERVER_TEMPLATE")
28 adminPort         = gp.getProperty("WEB_SERVER_ADMIN_PORT")
29 ihsAdminUserID    = gp.getProperty("WEB_SERVER_ADMIN_USERID")
30 ihsAdminPassword   = gp.getProperty("WEB_SERVER_ADMIN_PASSWORD")
31 pluginInstallRoot = gp.getProperty("PLUGIN_INSTALL_ROOT")
32
33
34 #Call the createWebServer() function.
35 #
36 createWebServer(nodeName, hostName, nodeOperatingSystem,
37                  webServerName, templateName, adminPort,
38                  ihsAdminUserID, ihsAdminPassword, pluginInstallRoot)
39

```

- \_\_\_ 2. Save `ut08_createWebServer.py`.

- \_\_\_ 3. Start the deployment manager if it is not already started.
  - \_\_\_ a. From a terminal window, go to <profile\_root>/Dmgr/bin.
  - \_\_\_ b. Enter the command: ./startManager.sh
  
- \_\_\_ 4. Run **ut08\_createWebServer.py** from a wsadmin shell.
  - \_\_\_ a. Go to <profile\_root>/Dmgr/bin.
  - \_\_\_ b. Enter the command: ./wsadmin.sh
  - \_\_\_ c. Enter the following command:  
  

```
execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject
/Scripts/ut08_createWebServer.py")
```
  - \_\_\_ d. If there are no errors in your scripts, you should see output similar to the following in the terminal window:



```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./wsadmin.sh
WASX7209I: Connected to process "dmgr" on node dmgrNode using SOAP connector; T
he type of process is: DeploymentManager
WASX7031I: For help, enter: "print Help.help()"
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut08
_createWebServer.py")
Creating ihsnode unmanaged node
ihsnode unmanaged node created
Creating Web server definition for webserver1 on ihsnode
Web server definition for webserver1 on ihsnode created
wsadmin>
```

- \_\_\_ e. Since the `AdminConfig.save()` line was commented out in the `ex08_createWebServer.py` script, type `AdminConfig.save()` at the wsadmin prompt to save the changes.



### Important

Make sure to run the `AdminConfig.save()` command; otherwise, you do not see the effect of your changes when you verify the results in the next section.

## Verifying the results

- \_\_\_ 1. First, verify that the *ihsnode* unmanaged node was properly created. Using the administrative console, verify that the two cluster members were created.
  - \_\_\_ a. Start the Firefox web browser, and enter the web address  
`http://was85host:9061/ibm/console`

- \_\_\_ b. Log in using **wasadmin** and **websphere** as the user ID and password.
- \_\_\_ c. In the navigation pane, select **System administration > Nodes**. The right pane shows the list of nodes that are defined in the cell configuration.

The screenshot shows the 'Nodes' management interface. At the top, there are buttons for 'Add Node', 'Remove Node', 'Force Delete', 'Synchronize', 'Full Resynchronize', and 'Stop'. Below this is a toolbar with icons for selecting, adding, removing, and synchronizing nodes. A table lists four nodes: 'PlantsNode01', 'PlantsNode02', 'dmgrNode', and 'ihsnode'. The 'ihsnode' row is selected, indicated by a red box around its entire row. The table columns include 'Select', 'Name', 'Host Name', 'Version', 'Discovery Protocol', and 'Status'. A note at the bottom says 'You can administer the following resources:' followed by the node list. A summary at the bottom states 'Total 4'.

Make sure that the list includes the unmanaged *ihsnode* that your script created.

- \_\_\_ d. Click the **ihsnode** link to view its properties.

The screenshot shows the 'Nodes > ihsnode' properties dialog. It has tabs for 'Configuration' and 'Local Topology', with 'Configuration' selected. Under 'General Properties', fields for 'Name' (ihsnode), 'Host Name' (was85host), and 'Platform Type' (Linux) are shown. These three fields are highlighted with a red box. On the right, sections for 'Node agent' (with 'Middleware agent' listed) and 'Additional Properties' (with 'Custom Properties' and 'Node Installation Properties' listed) are visible. At the bottom are 'Apply', 'OK', 'Reset', and 'Cancel' buttons.

The node name, host name, and platform type properties should match the values that you specified in the script.

- \_\_ 2. Next, verify that the web server was properly defined in the configuration. In the navigation pane, select **Servers > Server Types > Web Servers**.

The screenshot shows the 'Web servers' administrative interface. At the top, there's a toolbar with buttons for 'Generate Plug-in', 'Propagate Plug-in', 'New...', 'Delete', 'Templates...', 'Start', 'Stop', and 'Terminate'. Below the toolbar is a toolbar with icons for creating, deleting, and managing resources. A search bar at the top allows filtering by 'Name', 'Web server Type', 'Node', 'Host Name', 'Version', and 'Status'. A message below the search bar says 'You can administer the following resources:'. A table below lists one resource: 'webserver1' (selected), 'IBM HTTP Server', 'ihsnode', 'was85host', 'Not applicable', and a green 'Edit' icon. A total count of 1 is shown at the bottom.

The right pane shows an entry for *webserver1* in the displayed web servers list. In addition, it is configured on the *ihsnode*, and its type is *IBM HTTP Server*.

- \_\_ a. First, verify that the web server can be properly stopped and started from the administrative console. Select the check box next to **webserver1** and click the **Stop** button to stop the server.

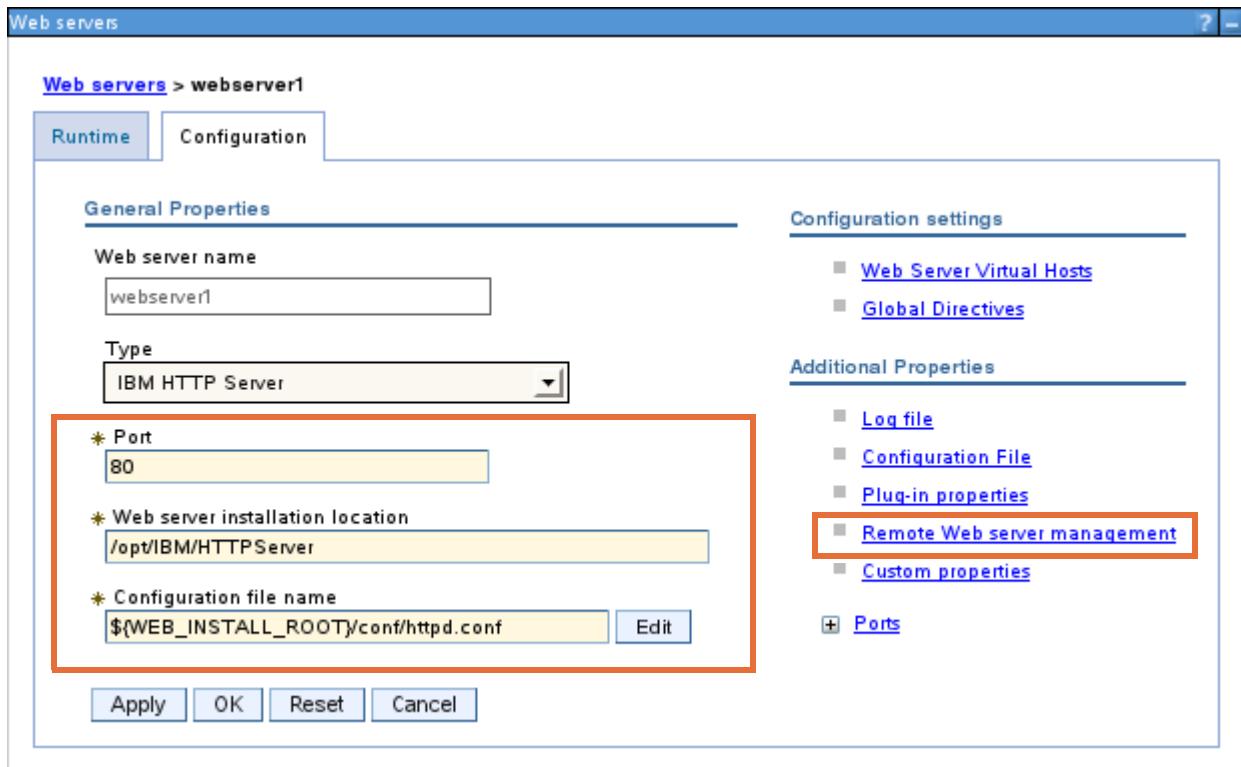


### Information

If a message is returned stating that the server cannot be stopped, or the console cannot determine the state of the server, it might indicate that WebSphere Application Server cannot communicate with the IBM HTTP Server administration service. This error is likely generated if the IBM HTTP Server administration service is not running. If the service is running, the error might be caused because the supplied IBM HTTP Server administration service user ID and password are incorrect.

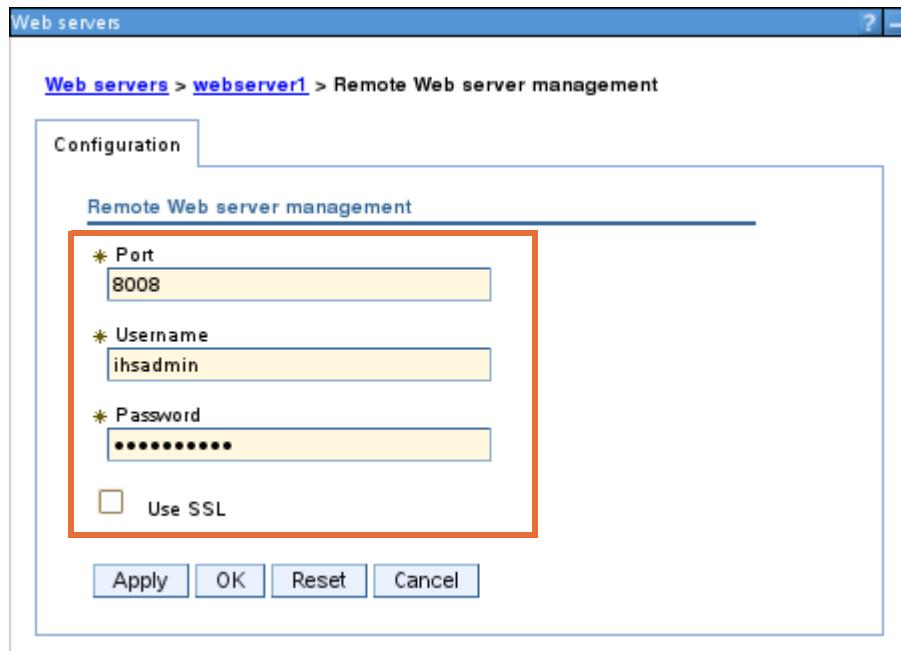
- \_\_ b. Select the check box again and click the **Start** button to start the web server.

- \_\_\_ c. To verify that the other web server properties are set correctly, click the **webserver1** link.



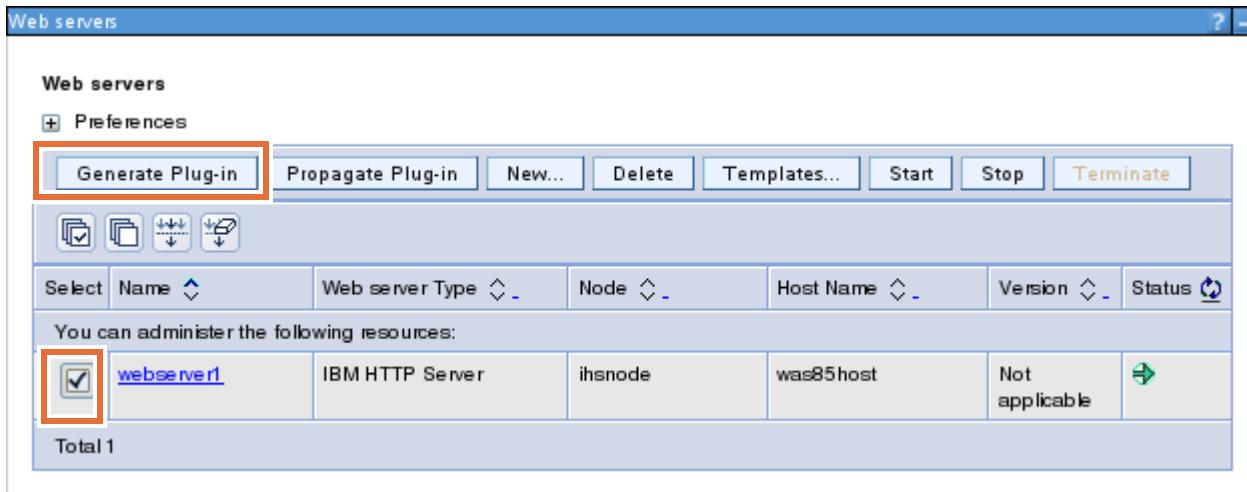
The configuration pane shows the general properties of the web server. They should correctly identify the port number, installation directory, and service name of the server.

- \_\_\_ d. Click the **Remote Web server management** link.

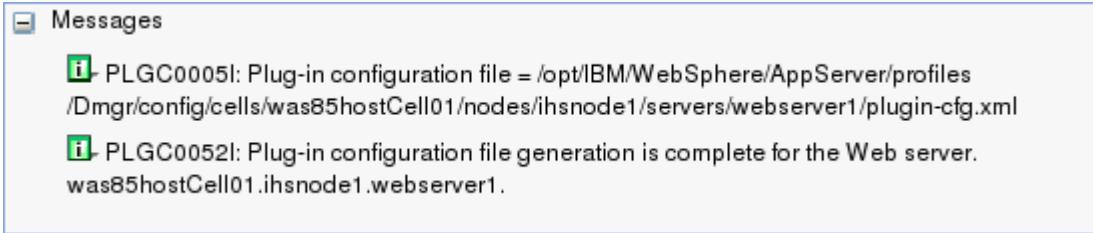


The values that are shown for Port, Username, and Password match those values that were supplied in the arguments for the `-remoteServerConfig` step of the `AdminTask.createWebServer()` method.

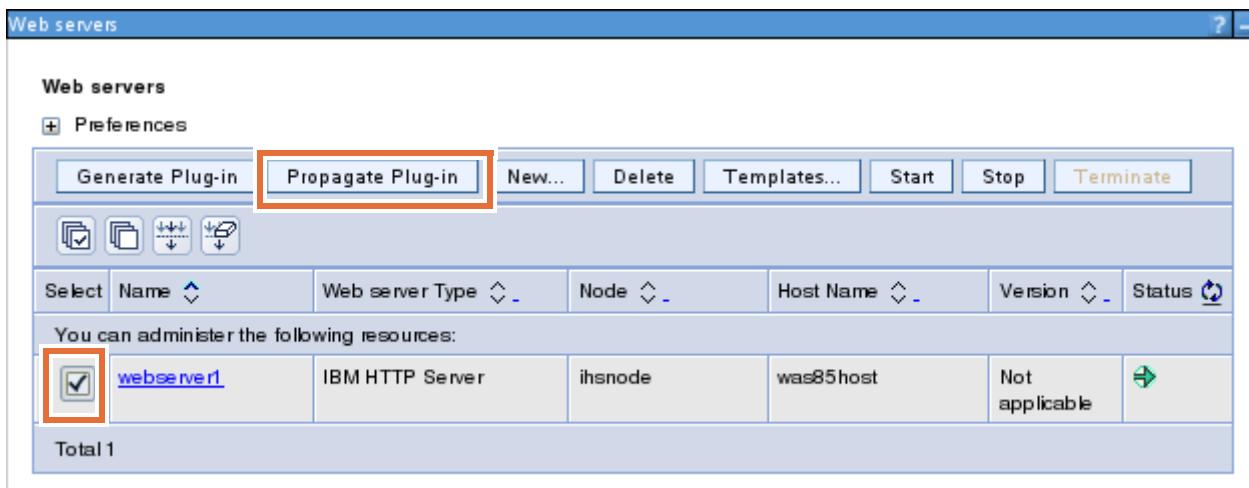
- \_\_\_ 3. Now that you verified that the web server was defined correctly, verify that the plug-in generation and propagation works. In the navigation pane, select **Servers > Server Types > Web servers**.
- \_\_\_ a. Select the check box beside the **webserver1** web server entry and click the **Generate Plug-in** button.



The following message is displayed, indicating that the plug-in was successfully generated for the `webserver1` web server. It also indicates the location of the generated `plugin-cfg.xml` file.



- \_\_\_ b. Select the check box beside the **webserver1** web server entry again, and click the **Propagate Plug-in** button.



The following message is displayed, indicating that the plug-in was successfully propagated to the web server machine. It also shows the location where the plugin-cfg.xml file was propagated.

Messages

- [i] PLGC0062I: The plug-in configuration file is propagated from /opt/IBM/WebSphere/AppServer/profiles/Dmgr/config/cells/was85hostCell01/nodes/ihsnode1/servers/webserver1/plugin-cfg.xml to /opt/IBM/WebSphere/Plugins/config/webserver1/plugin-cfg.xml on the Web server computer.
- [i] PLGC0048I: The propagation of the plug-in configuration file is complete for the Web server. was85hostCell01.ihsnode1.webserver1.



### Important

In the first message, notice where the plug-in configuration file is propagated to. In this case, /opt/IBM/WebSphere/Plugins/config/webserver1/plugin-cfg.xml. Verify that the **WebSpherePluginConfig** directive in the HTTP server's `httpd.conf` file also points to this location.



### Information

When a new plugin-cfg.xml file is propagated to the IBM HTTP Server, the web server loads any changes within 60 seconds.

- \_\_\_ 4. Log out of the administrative console.

You successfully created a Jython script to create an unmanaged node and define a web server.

## Section 4: Cleaning up the environment

- \_\_\_ 1. Stop the wsadmin shell by entering: quit.
- \_\_\_ 2. Stop the deployment manager server.
  - \_\_\_ a. From the terminal window, stop the deployment manager by entering the following command:  
`./stopManager.sh -username wasadmin -password web1sphere`
- \_\_\_ 3. Close any scripts that are still open in the Jython editor.
- \_\_\_ 4. Exit the IBM Assembly and Deploy Tools by selecting **File > Exit**.

**End of exercise**

## Exercise review and wrap-up

In this exercise, you learned how to use the IBM Installation Manager to record a response file for installing the IBM HTTP Server and the web server plug-ins. Then, you installed the IBM HTTP Server and its plug-in by using a silent installation. Finally, you developed Jython scripts to configure the IBM HTTP Server as an unmanaged node.

# Exercise 9. Deploying the PlantsByWebSphere application

## What this exercise is about

In this exercise, you use Jython scripts to deploy and manage the PlantsByWebSphere application. You also use scripts to configure environment variables, a data source, and other resources that the PlantsByWebSphere application uses.

## What you should be able to do

At the end of this exercise, you should be able to develop Jython scripts capable of:

- Set WebSphere environment variables
- Create database resources, including J2C authentication aliases, JDBC providers, and data sources
- Install an enterprise application to a cluster
- Map application modules to servers

## Introduction

Figure 1 highlights the application and resources that the PlantsByWebSphere application uses, and that you create and configure in this exercise.

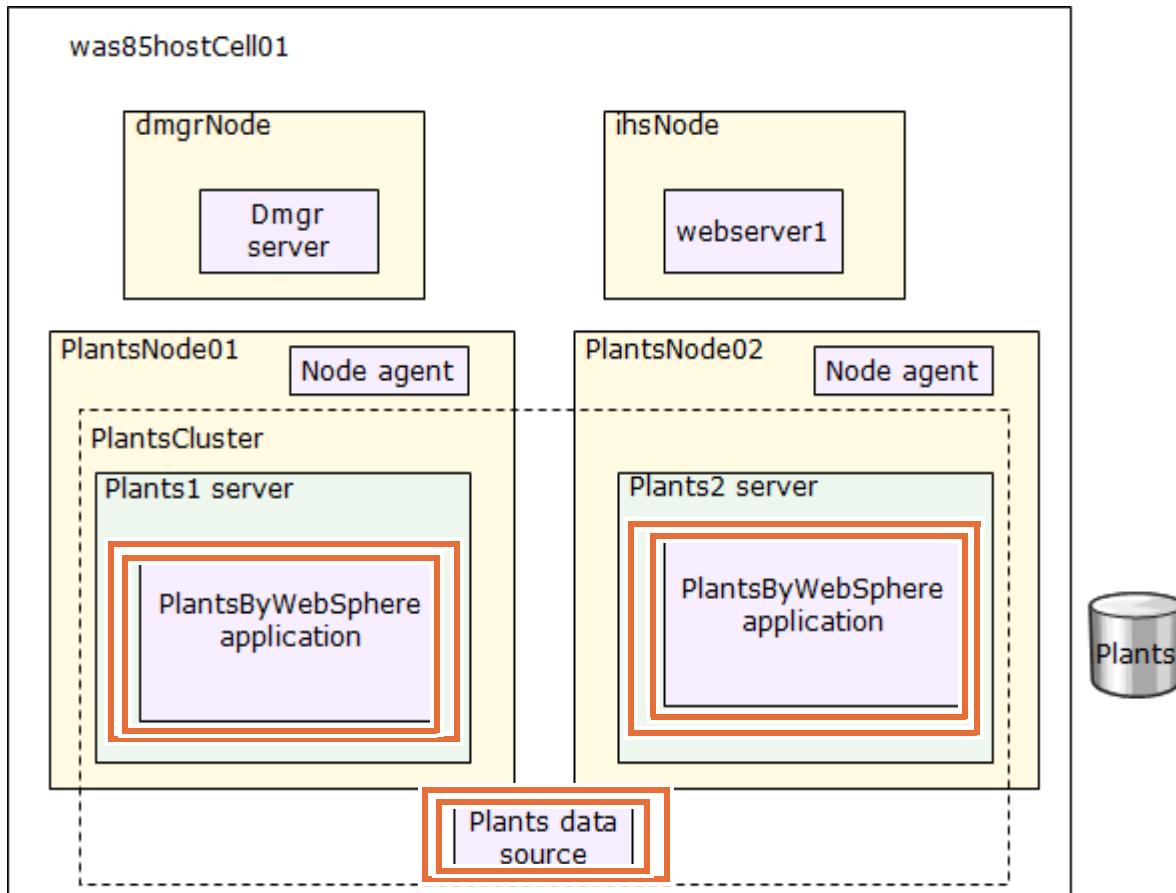


Figure 1 - Plants deployment environment servers and server resources

As you can see, in this exercise you complete the configuration of the resources that the PlantsByWebSphere application requires, and install the application.

## Requirements

To complete this exercise, you must have the WebSphere Application Server Network Deployment V8.5 product installed. You also require that the IBM Assembly and Deploy Tools for WebSphere Administration V8.5. This exercise requires the completion of the previous two exercises in which you install the deployment manager profile, two managed profiles, the IBM HTTP Server, and WebSphere plug-in. You also created the PlantsCluster, two cluster members, and configured the web server in the cell configuration.



## Exercise instructions



### Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

`<was_root>`: /opt/IBM/WebSphere/AppServer

`<profile_root>`: /opt/IBM/WebSphere/AppServer/profiles

### Section 1: Setting up the environment

#### Preparing the PlantsByWebSphere application files for installation

This section of the exercise copies the EAR files that contain the PlantsByWebSphere application to the folder where they are installed from. Typically this folder is the *installableApplications* folder under the profile of the server where you are installing the applications. For this exercise, this profile is the deployment manager's profile.

- \_\_\_ 1. Copy the PlantsByWebSphere application EAR file to the installableApps folder of the **Dmgr** profile.
  - \_\_\_ a. Copy the following EAR file from the `/usr/Software/Ears` folder to the `<profile_root>/Dmgr/installableApps` directory
    - PlantsByWebSphere.ear



### Information

EAR files are typically placed under the

`<profile_root>/<profile_name>/installableApps` directory. This directory is a standard convention but by no means a requirement. The enterprise application path is specified at the time of installation and can be set to a directory of your choosing.

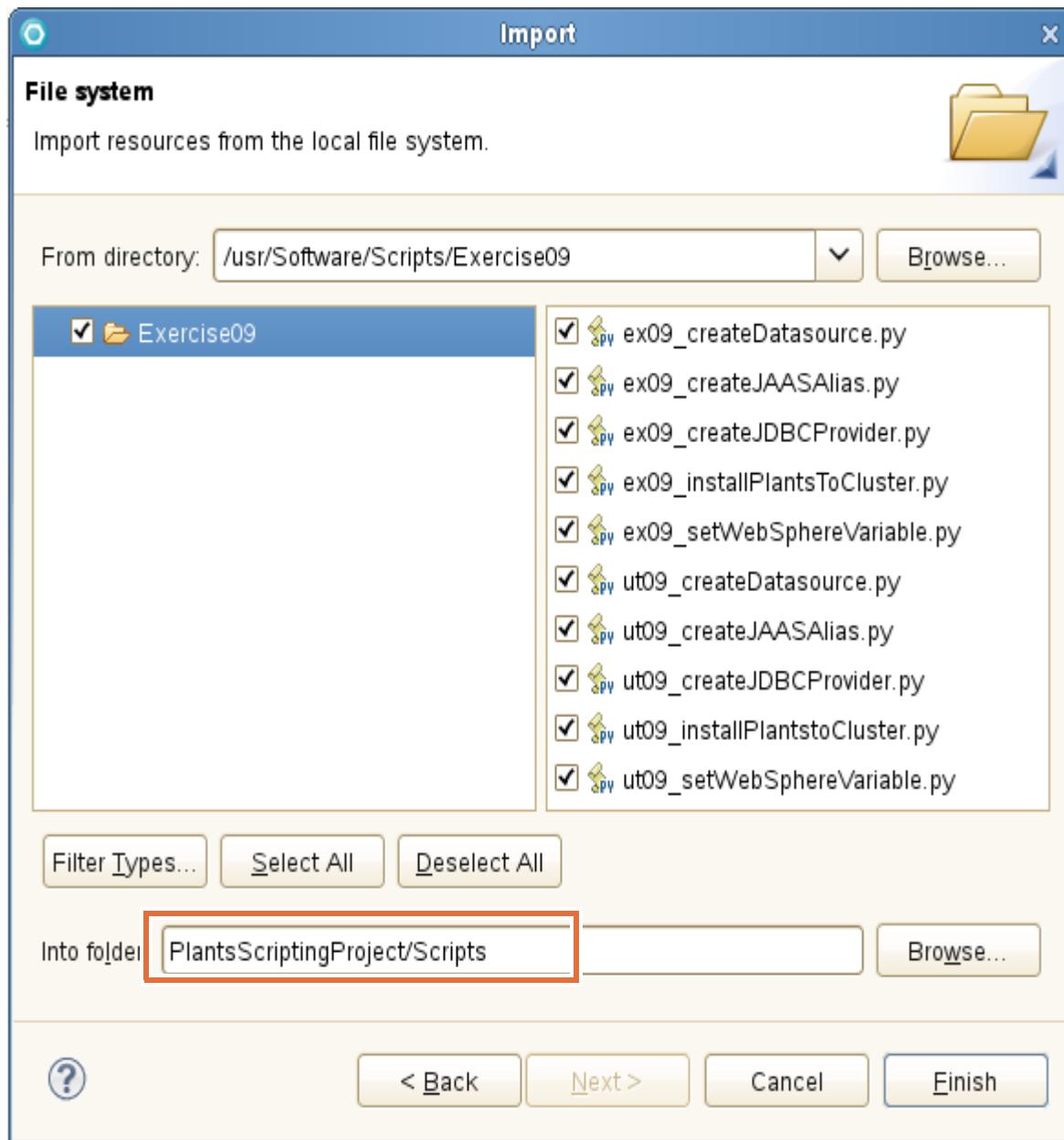
#### Make sure IBM Assembly and Deploy Tools and the deployment manager are up running

- \_\_\_ 1. If not already running, start IBM Assembly and Deploy Tools (IADT) and open to the `/usr/LabWork/DistributedWS` workspace.
- \_\_\_ 2. If not already running, start the deployment manager.
  - \_\_\_ a. From a Terminal window, go to `<profile_root>/Dmgr/bin`
  - \_\_\_ b. Enter the command: `./startManager.sh`

## Import startup scripts

- \_\_\_ 1. Import the skeleton function and mainline scripts that are designed to help you in your script development.
  - \_\_\_ a. In the Enterprise Explorer view, right-click **PlantsScriptingProject** and select **Import**.
  - \_\_\_ b. In the *Import select* dialog, expand **General** and select **File system**.
  - \_\_\_ c. Click **Next**.
  - \_\_\_ d. In the *File system* dialog, click **Browse** next to the **From** directory field.
  - \_\_\_ e. In the *Import from directory* dialog, go to **/usr/Software/Scripts**, select **Exercise09**, and click **OK**.
  - \_\_\_ f. In the *File system* dialog:
    - Select the **Exercis09** check box to select all of the files in the folder:
      - ex09\_createDatasource.py
      - ex09\_createJAASAlias.py
      - ex09\_createJDBCProvider.py
      - ex09\_installPlantsToCluster.py
      - ex09\_setWebSphereVariable.py
      - ut09\_createDatasource.py
      - ut09\_createJAASAlias.py
      - ut09\_createJDBCProvider.py
      - ut09\_installPlantsToCluster.py
      - ut09\_setWebSphereVariable.py

- g. Type **/Scripts** at the end of **PlantsScriptingProject** in the Into folder field. This action causes the files to be imported in a folder named *Scripts* under the project.



- h. Click **Finish**.
- i. In the Enterprise Explorer view, expand **PlantsScriptingProject > Scripts**. The skeleton function and mainline scripts for this exercise are in the folder.

## Section 2: Create resources for the PlantsByWebSphere application

Before you proceed to create the scripts that install the PlantsByWebSphere application, you must set up the many resources needed for the applications to function in the lab environment. Each section in this part of the exercise creates and tests a set of resources.

### Create WebSphere environment variables

In this section, you create a function that can create or update any of the WebSphere environment variables, although in this exercise you use this function to update only the DB2 driver path variables. You also create a unit test driver to test the function.



#### Information

Since you do not need the node agents and the servers in the cluster for a while, you might want to stop them to free up memory and system resources. Stopping these servers improves overall image performance during this exercise.

There are two WebSphere variables that are used when defining JDBC providers. These variables point to the location of the database drivers, in the context of this exercise the DB2 JDBC drivers.

Your specific requirements are as follows:

1. Create a Jython function named `setWebSphereVariable` in a file called `ex09_setWebSphereVariable.py`. This function should be able to create or update WebSphere variables. The following parameters are passed to the function:
  - `nodeName` - name of node that represents the scope of the variable
  - `envVarName` - name of variable to be created or updated
  - `envVarValue` - value of the variable
2. Create a unit test driver, `ut09_setWebSphereVariable.py`, which sets up the necessary constants and parameters to invoke the `setWebSphereVariable()` function to update the `DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH` WebSphere variable on the first node, **PlantsNode01**, scope.
3. Still, in the unit test driver, invoke the function again for the `DB2UNIVERSAL_JDBC_DRIVER_PATH` WebSphere variable for the same scope.
4. Repeat for both variables by using the second node's scope, **PlantsNode02**.
- \_\_\_ 1. Complete the `ut09_setWebSphereVariable.py` unit test script.



## Information

In many cases it is considered best practice to write the unit test for a function before writing the actual function. Doing so gives you a flavor for how you use the function without regard for the implementation of the function.

- \_\_ a. In IADT, open the `ut09_setWebSphereVariable.py` unit test script.
- \_\_ b. Review the partially completed unit test script.

```

1 # ut09_setWebSphereVariable.py
2
3 # Load the PropertyLoaderClass and the script function to be called
4 execfile(UTIL_LOCATION + "/PropertyLoaderClass.py")
5 execfile(SCRIPT_LOCATION + "/ex09_setWebSphereVariable.py")
6
13 gp=PropertyLoaderClass()
14
15 # Write code to test function below
16
17 # Get required properties
18 db2DriverPath = gp.getProperty("")
19 db2DriverNativePath = gp.getProperty("")
20 # Need to remove the single quotes enclosing the DB2 class path
21 db2DriverClasspath = gp.getProperty("DB2_DRIVER_CLASSPATH").replace("'", "")
22
23 # Get PlantsNode01 name
24 node1Name = gp.getProperty("")
25
26 # Invoke function to set WebSphere variable for PlanysNode01
27 setWebSphereVariable()
28 setWebSphereVariable()
29
30
31 # Get PlantsNode02 name
32 node2Name = gp.getProperty("")
33
34 # Invoke function to set WebSphere variable for PlantsNode02
35 setWebSphereVariable()
36 setWebSphereVariable()

```

- \_\_ c. Retrieve the required constants from the properties file by using the `gp` instance of the property loader class. Here is a list of the required properties:

- DB2\_DRIVER\_PATH
- DB2\_DRIVER\_NATIVEPATH
- DB2\_DRIVER\_CLASSPATH
- PLANTS\_NODE01\_NAME
- PLANTS\_NODE02\_NAME

- \_\_\_ d. Complete the calls to **setWebSphereVariable()** passing parameters appropriate to fulfill the specifications that are given.

**Hint**

The parameters that are required for each call are the node name, the variable name, and the variable's value. See the function's signature in the following step.

**Information**

Constants that contain spaces are enclosed in single quotes in the `plants.properties` file. The use of single quotes makes them easier to use in the scripts because you do not have to use string concatenation to add the quotes before you can use the variables.

However, the single quotes can get in the way when the constants are used in comparisons, either in your code or by code in library functions. In this case, the single quotes must be removed before using the constant.

A simple way to remove the single quotes is to use the **replace()** method of the string object as shown on line 21 of `ut09_setWebSphereVariable.py`.

```
db2DriverClasspath = gp.getProperty("DB2_DRIVER_CLASSPATH").replace("'", "")
```

Getting the classpath from the properties file is the same as you did in the past. Once you have the string that represents the provider, you invoke the **replace()** method of the string class, this method looks for the first character that is passed in and replaces it with the second parameter. In this case, the second parameter is an empty string, which results in replacing the single quote with nothing, in effect it removes all single quotes from the string.

Review the code on line 21 and make sure that you understand how single quotes are being removed.

- \_\_\_ e. Save your changes. You test the script later after the script that contains the function is completed.
- \_\_\_ 2. Complete the `ex09_setWebSphereVariable.py` script.
- \_\_\_ a. Open the **ex09\_setWebSphereVariable.py** script.

- \_\_ b. Review the partially completed `setWebSphereVariable()` function.

```

SPV ex09_SetWebSphereVariable.py ✎
1 # ex09_SetWebSphereVariable.py
2 def setWebSphereVariable(nodeName, envVarName, envVarValue):
3     print "Setting WebSphere variable:", envVarName, "to value:", envVarValue
4
5     # Get name of cell using AdminControl
6     cell = AdminControl.getCell()
7
8     # Create scope variable
9     scope = "Cell=" + cell + ",Node=" + nodeName
10
11    # Create envVarOpts variable containing variableName, variableValue and scope
12    envVarOpts = ["-variableName", envVarName, "-variableValue", envVarValue, "-scope", scope]
13
14    # Use AdminTask to set the WebSphere variable
15    AdminTask.setVariable(envVarOpts)
16
17    # Save the changes
18    AdminConfig.save()
19
20    print "Saved WebSphere variable:", envVarName, "with value:", envVarValue, "to configuration"

```

The `setWebSphereVariable()` method accepts the following parameters:

- `nodeName` - specifies the name of the node onto which the WebSphere variable is changed.
  - `envVarName` - the WebSphere environment variable name
  - `envVarValue` - specifies the value to be assigned to the WebSphere environment variable.
- \_\_ c. The first thing the script needs to obtain is the name of the WebSphere cell. You obtained the cell name before by using the `getCell()` method of the `AdminConfig` object. Assign the name of the cell to the `cell` variable.
- \_\_ d. The next step hints to creating a scope variable. In this case, the scope is made up of two name value pairs, by using the cell and node names.



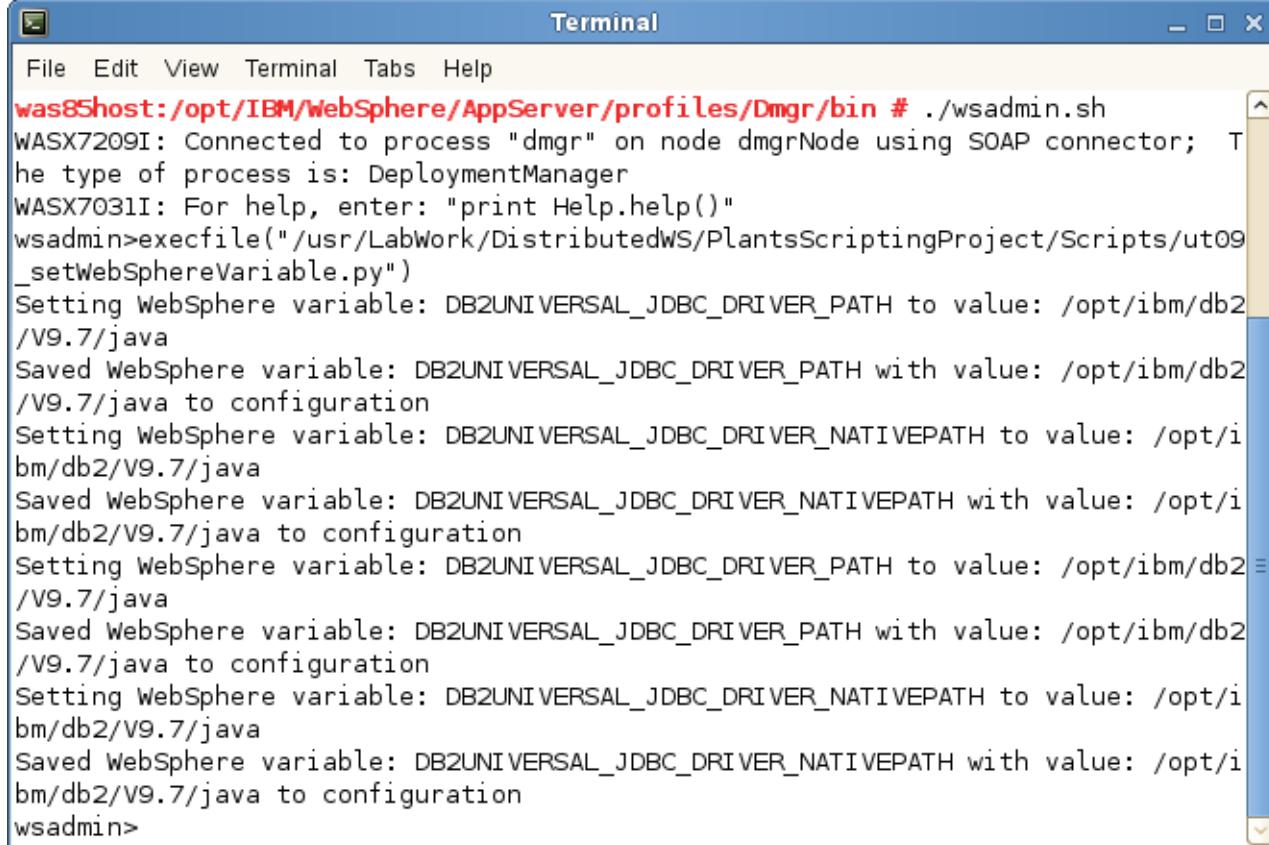
### Hint

By looking at the code segment above, you notice that this function ultimately uses the `AdminTask` object to operate on WebSphere variables. As you might remember from the lecture, one of the command groups for the `AdminTask` object is `VariableConfiguration`. Sounds like this command group might be useful since you are trying to configure a WebSphere variable. In the WebSphere Information Center search for the following keywords: `AdminTask variableConfiguration`.

From this help item, you can determine which method of `AdminTask` can be used to set the variable and what parameters are required, as well as the format of the parameters.

Initialize the variable scope to the proper value by using the proper format of the cell and node names.

- \_\_\_ e. Assign the proper value to the **envVarOpts** variable. This variable holds a list that contains name/value pairs that represent the variable name, the variable's value and the scope of the variable.
  - \_\_\_ f. Use the **setVariable()** method of AdminTask to change the variable.
  - \_\_\_ g. Save the changes to the configuration.
  - \_\_\_ h. Save the script file in IADT.
- \_\_\_ 3. Run **ut09\_setWebSphereVariable.py** from a wsadmin shell.
- \_\_\_ a. Navigate to <profile\_root>/Dmgr/bin.
  - \_\_\_ b. Enter the command: ./wsadmin.sh
  - \_\_\_ c. Enter the following command:
- ```
execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut09_setWebSphereVariable.py")
```



The screenshot shows a terminal window titled "Terminal". The window has a menu bar with File, Edit, View, Terminal, Tabs, Help. The main area displays the output of a command-line session:

```

File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./wsadmin.sh
WASX7209I: Connected to process "dmgr" on node dmgrNode using SOAP connector; T
he type of process is: DeploymentManager
WASX7031I: For help, enter: "print Help.help()"
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut09_setWebSphereVariable.py")
Setting WebSphere variable: DB2UNIVERSAL_JDBC_DRIVER_PATH to value: /opt/ibm/db2/V9.7/java
Saved WebSphere variable: DB2UNIVERSAL_JDBC_DRIVER_PATH with value: /opt/ibm/db2/V9.7/java to configuration
Setting WebSphere variable: DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH to value: /opt/ibm/db2/V9.7/java
Saved WebSphere variable: DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH with value: /opt/ibm/db2/V9.7/java to configuration
Setting WebSphere variable: DB2UNIVERSAL_JDBC_DRIVER_PATH to value: /opt/ibm/db2/V9.7/java
Saved WebSphere variable: DB2UNIVERSAL_JDBC_DRIVER_PATH with value: /opt/ibm/db2/V9.7/java to configuration
Setting WebSphere variable: DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH to value: /opt/ibm/db2/V9.7/java
Saved WebSphere variable: DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH with value: /opt/ibm/db2/V9.7/java to configuration
wsadmin>
```

- \_\_\_ d. Examine the output from this command in the terminal window. Correct any errors that might show up and rerun until successful.
- \_\_\_ e. Leave the wsadmin shell open in the terminal window. You are going to use it several more times during this exercise.

**Information**

This script does not produce any side effects if it is run more than once, as the same variables get written over.

- \_\_\_ 4. Verify the WebSphere Environment variables are updated.
  - \_\_\_ a. Log on to the WebSphere administrative console.
  - \_\_\_ b. From the navigator pane, select **Environment > WebSphere variables**.
  - \_\_\_ c. Select **All scopes**.
  - \_\_\_ d. Scroll through the list of environment variables, and verify that the DB2UNIVERSAL\_JDBC\_DRIVER\_NATIVEPATH and DB2UNIVERSAL\_JDBC\_DRIVER\_PATH are correctly set to  
**/opt/ibm/db2/v9.7/java**  
for both the PlantsNode01 and PlantsNode02 nodes.

<input type="checkbox"/>	<a href="#">DB2UNIVERSAL JDBC DRIVER_NATIVEPATH</a>	/opt/ibm/db2/V9.7/java	Node=PlantsNode01
<input type="checkbox"/>	<a href="#">DB2UNIVERSAL JDBC DRIVER_NATIVEPATH</a>	/opt/ibm/db2/V9.7/java	Node=PlantsNode02
<input type="checkbox"/>	<a href="#">DB2UNIVERSAL JDBC DRIVER_PATH</a>		Node=dmgrNode
<input type="checkbox"/>	<a href="#">DB2UNIVERSAL JDBC DRIVER_PATH</a>	/opt/ibm/db2/V9.7/java	Node=PlantsNode01
<input type="checkbox"/>	<a href="#">DB2UNIVERSAL JDBC DRIVER_PATH</a>	/opt/ibm/db2/V9.7/java	Node=PlantsNode02

- \_\_\_ e. Log out of the administrative console.

## Create the JAAS aliases

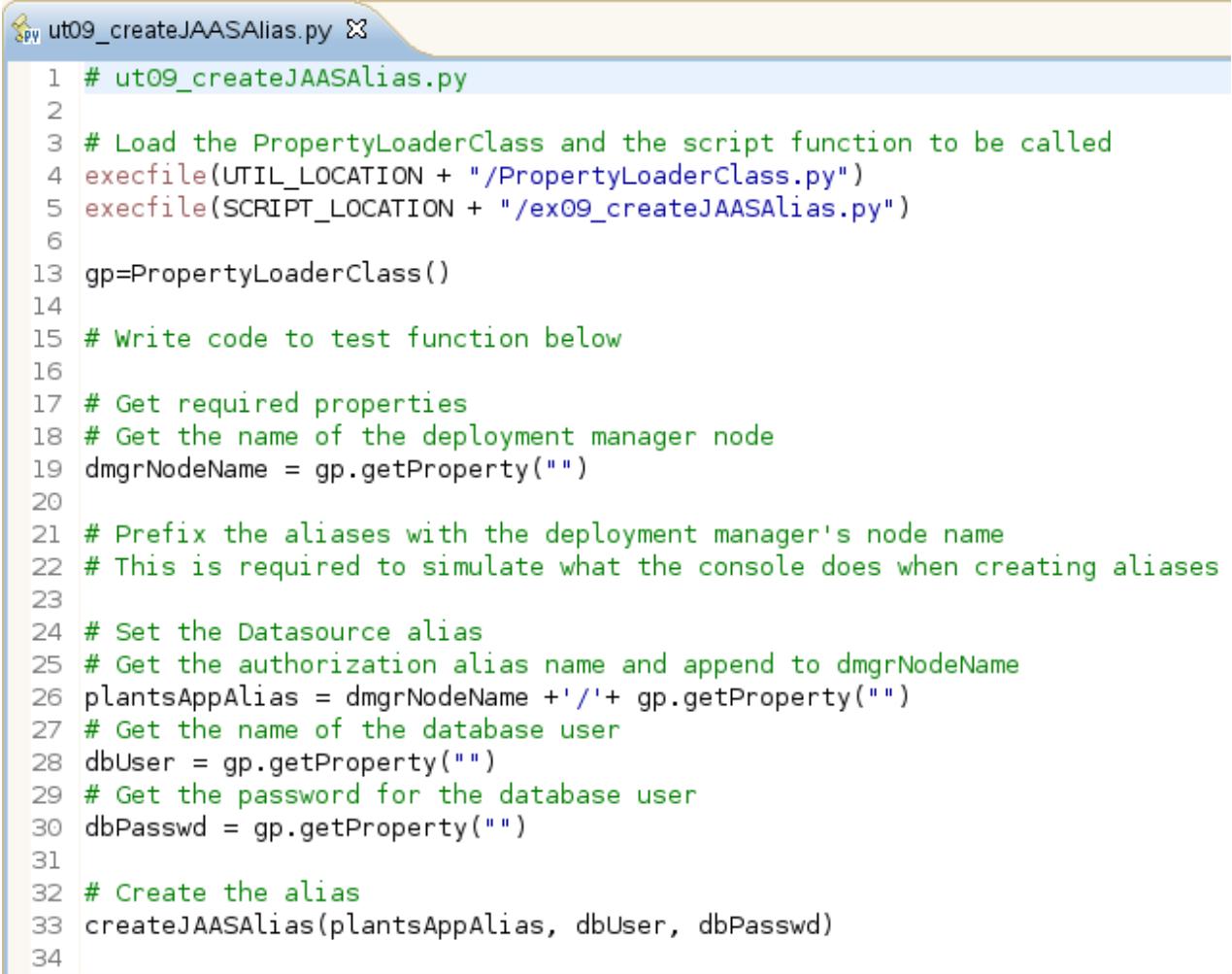
In this part, you create a function that can create JAAS aliases. WebSphere defined resources, such as data sources, use JAAS, or J2C, aliases when accessing a secured component, such as a database server. These objects contain an alias name, a user id, and a password, which the resource recognizes. You also create a unit test driver to test the function.

There is one JAAS alias that the resources of the PlantsByWebSphere application require. This alias is used when the data sources access the databases.

Your specific requirements are as follows:

1. Create a Jython function named `createJAASAlias` in a file that is called `ex09_createJAASAlias.py` to create an authentication alias. The following parameters are passed to the function:

- nodeName - name of node that represents the scope of the variable
  - envVarName - name of variable to be created or updated
  - envVarValue - value of the variable
2. Create a unit test driver, ut09\_createJAASAlias.py, which sets up the necessary constants and parameters to invoke the `createJAASAlias()` function to create the JAAS alias: PlantsApp.
- \_\_\_ 1. Complete the `ut09_createJAASAlias.py` unit test script.
- \_\_\_ a. Edit the **ut09\_createJAASAlias.py** unit test script in the Jython editor of IADT.
- \_\_\_ b. Review the partially completed unit test script



```

1 # ut09_createJAASAlias.py
2
3 # Load the PropertyLoaderClass and the script function to be called
4 execfile(UTIL_LOCATION + "/PropertyLoaderClass.py")
5 execfile(SCRIPT_LOCATION + "/ex09_createJAASAlias.py")
6
13 gp=PropertyLoaderClass()
14
15 # Write code to test function below
16
17 # Get required properties
18 # Get the name of the deployment manager node
19 dmgrNodeName = gp.getProperty("")
20
21 # Prefix the aliases with the deployment manager's node name
22 # This is required to simulate what the console does when creating aliases
23
24 # Set the Datasource alias
25 # Get the authorization alias name and append to dmgrNodeName
26 plantsAppAlias = dmgrNodeName +'/' + gp.getProperty("")
27 # Get the name of the database user
28 dbUser = gp.getProperty("")
29 # Get the password for the database user
30 dbPasswd = gp.getProperty("")
31
32 # Create the alias
33 createJAASAlias(plantsAppAlias, dbUser, dbPasswd)
34

```

- \_\_\_ c. In the appropriate line of code, retrieve the required constants from the properties file by using the `gp` instance of the property loader class. Here is a list of the required properties:

-DMGR\_NODE\_NAME  
 -AUTH\_ALIAS\_NAME

-DB\_USER  
-DB\_PASSWORD

- \_\_\_ d. Create the complete authorization alias name for the database by prepending the deployment manager's node name followed by a forward slash to the alias name.



**Note**

When JAAS aliases are created by using the WebSphere administrative console, the deployment manager's node name is prepended to the name of the alias. In the unit test script, you simulate this side effect in the code.

- \_\_\_ e. Call the function to create the alias.



**Hint**

The parameters that are required for each call are the alias name, the user id, and the password.

- \_\_\_ f. Save your changes. You test the script later after the script that contains the function is completed.
- \_\_\_ 2. Complete the **ex09\_createJAASAlias.py** script.
- \_\_\_ a. Open the **ex09\_createJAASAlias.py** script in the Jython editor.

- \_\_\_ b. Review the partially completed *createJAASAlias()* function.

```


  1 # ex09_createJAASAlias.py ✘
  2 def createJAASAlias(aliasName, user, passw):
  3     print "Creating JAAS alias: " + aliasName
  4
  5     # Get name of cell using AdminControl
  6     cell = AdminControl.getCell()
  7
  8     # Get configuration ID for the parent of the JAAS alias, Security
  9     security = AdminConfig.getId("")
10
11     # Create and initialize variables used as
12     # new alias parameters
13
14     alias=[]
15     userid=[]
16     password=[]
17
18     # Assemble the above variables into a list
19     jaasAttrs=[alias, userid, password]
20
21     try:
22         # Use AdminConfig to create the JAAS alias
23         AdminConfig.create()
24
25         # Save the changes
26         AdminConfig.save()
27
28         print "Created JAAS alias: " + aliasName
29     except:
30         print "Could not create JAAS alias: " + aliasName
31         print "possibly because it already exists..."

```

The *createJAASAlias()* method accepts the following parameters:

- aliasName - specifies the JAAS Alias named associated with the user and password
- user - the user name
- passw - the password for the specified user

- \_\_\_ c. Retrieve the *Security* configuration ID for the cell. This configuration ID is the parent object for the JAAS authorization aliases.



## Hint

Search the Information Center for: create JAAS alias scripting. Remember that JAAS aliases are also called Java 2 Connector (J2C) authentication aliases.

- \_\_\_ d. Create and populate the **alias**, **userid**, and **password** variables with appropriate name/value pairs.
  - \_\_\_ e. Create and populate the **jaasAttrs** variable with a list of the name/value pairs that are listed.
  - \_\_\_ f. The next step requires you to catch possible exceptions that are thrown if you attempt to create an alias that exists. Enclose the creation of the alias in a **try/except** block.
  - \_\_\_ g. Use the **create()** method of the AdminConfig object to create a **JAASAuthData** entry, the Security configuration ID, and the jaasAttrs are also parameters for the method call.
  - \_\_\_ h. Save the changes.
- \_\_\_ 3. Run **ut08\_createJAASAlias.py** from the wsadmin shell.

- \_\_\_ a. Enter the following command:

```
execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject
/Scripts/ut09_createJAASAlias.py")
```

The screenshot shows a terminal window with the title "Terminal". The window contains the following text:

```
File Edit View Terminal Tabs Help
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut09
_createJAASAlias.py")
Creating JAAS alias: dmgrNode/PlantsApp
Created JAAS alias: dmgrNode/PlantsApp
wsadmin>
```

- \_\_\_ b. Examine the output from this command in the terminal window. Correct any errors that might show up and rerun until successful.
  - \_\_\_ c. Uncomment the **AdminConfig.save()** line in the script and run one more time to create and save the JAAS alias.
- \_\_\_ 4. After the script runs once to completion, the alias is created. Run the script again to see how the exception handling works.
- \_\_\_ 5. Verify the JAAS aliases were created.
- \_\_\_ a. Log on to the WebSphere administrative console.
  - \_\_\_ b. From the navigation pane, select **Security > Global security**

- \_\_\_ c. Under Authentication, expand Java Authentication and Authorization Service and click J2C authentication data



- \_\_\_ d. Verify the dmgrNode/PlantsApp alias is created.

Select	Alias	User ID	Description
<input type="checkbox"/>	dmgrNode/PlantsApp	db2inst1	

The alias is successfully configured.

- \_\_\_ e. If you do not see the alias, run AdminConfig.save() from the wsadmin shell, and refresh the administrative console.
- \_\_\_ f. Log out of the WebSphere administrative console.

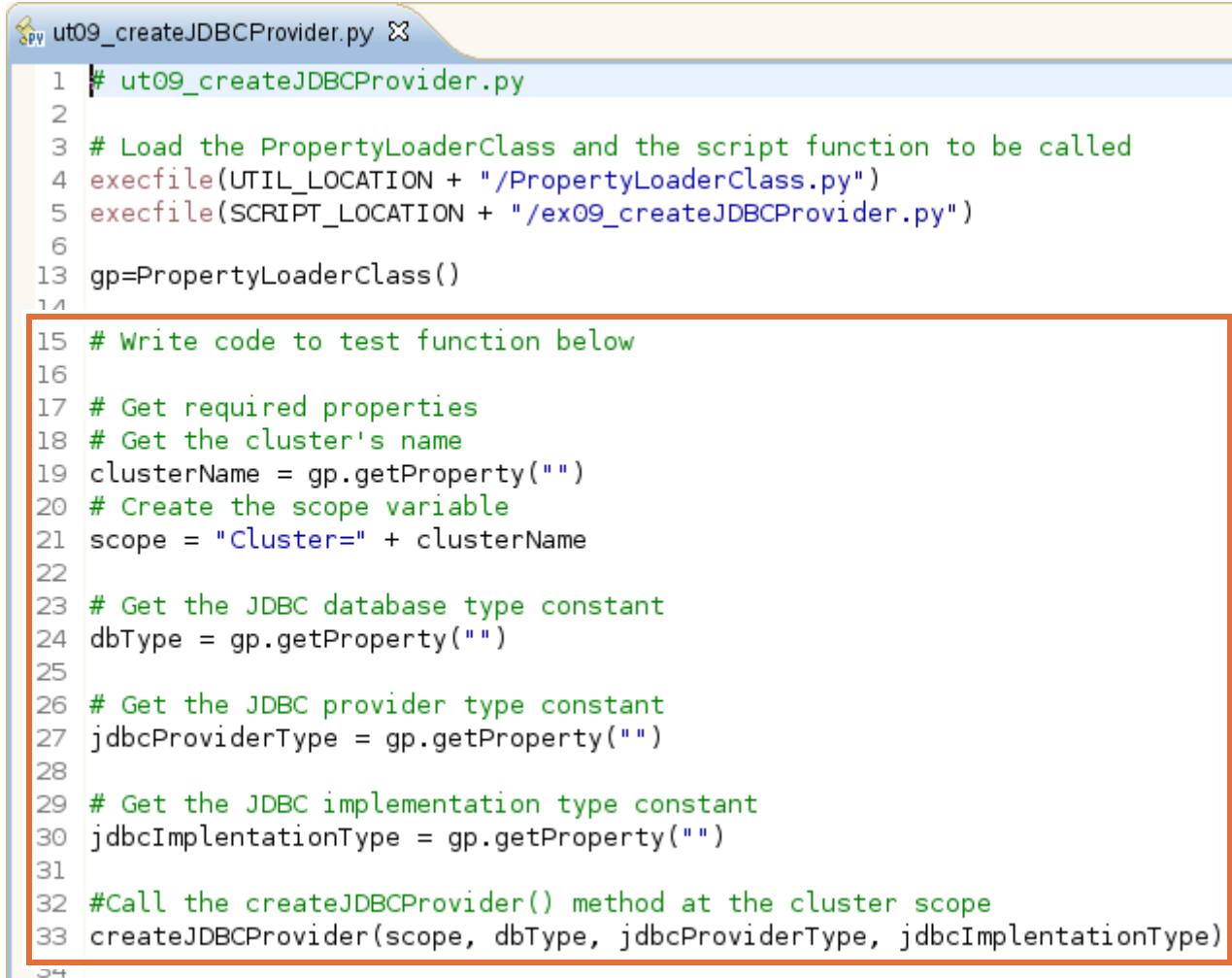
## Create a DB2 XA JDBC Provider

In this part, you create a function that can create JDBC providers. JDBC providers are used in WebSphere to supply connection pooled data sources for applications to use to access databases. You also create a unit test driver to test the function.

Your specific requirements are as follows:

1. Create a Jython function named `createJDBCProvider` in a file that is called `ex09_createJDBCProvider.py` to create a JDBC provider. The following parameters are passed in to the function:
  - `scope` - the scope in which to create the JDBC provider
  - `jdbcDBType` - the type of database the provider uses
  - `jdbcProviderType` - the provider type
  - `jdbcImplementationType` - whether this provider is a connection pool or XA data source provider
2. Create a unit test driver, `ut09_createJDBCProvider.py`, which sets up the necessary constants and parameters to invoke the `createJDBCProvider()` function.
  - \_\_\_ 1. Complete the `ut09_createJDBCProvider.py` unit test script.
  - \_\_\_ a. Edit the **`ut09_createJDBCProvider.py`** unit test script in the Jython editor.

- \_\_ b. Review the partially completed unit test script



```

1 # ut09_createJDBCProvider.py
2
3 # Load the PropertyLoaderClass and the script function to be called
4 execfile(UTIL_LOCATION + "/PropertyLoaderClass.py")
5 execfile(SCRIPT_LOCATION + "/ex09_createJDBCProvider.py")
6
7 gp=PropertyLoaderClass()
8
9
10 # Write code to test function below
11
12
13 # Get required properties
14 # Get the cluster's name
15 clusterName = gp.getProperty("")
16 # Create the scope variable
17 scope = "Cluster=" + clusterName
18
19 # Get the JDBC database type constant
20 dbType = gp.getProperty("")
21
22 # Get the JDBC provider type constant
23 jdbcProviderType = gp.getProperty("")
24
25 # Get the JDBC implementation type constant
26 jdbcImplementationType = gp.getProperty("")
27
28 # Call the createJDBCProvider() method at the cluster scope
29 createJDBCProvider(scope, dbType, jdbcProviderType, jdbcImplementationType)
30
31
32
33
34

```

- \_\_ c. In the appropriate lines of code, retrieve the required constants from the properties file by using the **gp** instance of the property loader class. Here is a list of the required properties that are defined in the `plants.properties` file:

- CLUSTER\_NAME
- JDBC\_DATABASE\_TYPE
- JDBC\_PROVIDER\_TYPE
- JDBC\_IMPLEMENTATION\_TYPE



### Information

In this unit driver, the variables that are listed are set to create an XA DB2 Universal JDBC provider. The function to create the driver should be flexible enough to create any supported JDBC provider that is based on the parameters passed.

- \_\_\_ d. After obtaining the cluster name, notice that the **scope** variable is created by adding `Cluster=` in front of the cluster name.



### Hint

To find out the format of the scope, consult the Information Center by searching for: `AdminTask createJDBCprovider`. As you might remember from the lecture, **JDBCProviderManagement** is one of the command groups for the AdminTask object. This page on the Information Center can also help you composing the command to actually create the JDBC provider in the next step.

- \_\_\_ e. Call the function to create the JDBC provider.
- \_\_\_ f. Save your changes. You test the script later after the script that contains the function is completed.
- \_\_\_ 2. Complete the `ex09_createJDBCProvider.py` script.
- \_\_\_ a. Open the **ex09\_createJDBCProvider.py** script in the Jython editor.
- \_\_\_ b. Review the partially completed `createJDBCProvider()` function.

```

py ex09_createJDBCProvider.py ✘
1 # ex09_createJDBCProvider.py
2 def createJDBCProvider(scope, jdbcDBType, jdbcProviderType, jdbcImplementationType):
3     print "Creating JDBC provider"
4
5     #Create scope variable
6     scope = "-scope " + scope
7
8     # Create database type variable
9     dbType = " -databaseType " + jdbcDBType
10
11    # Create jdbc provider type variable
12    providerType = " -providerType " + jdbcProviderType
13
14    # Create JDBC implementation type variable
15    implementationType = " -implementationType " + jdbcImplementationType
16
17    # Concatenate all attributes
18    jdbcAttrs = scope + dbType + providerType + implementationType
19    print "JDBC provider attributes: " + jdbcAttrs
20
21    # Call AdminTask to create the JDBC provider
22    AdminTask.createJDBCProvider(jdbcAttrs)
23
24    # Save configuration
25    AdminConfig.save()
26
27    print "Created JDBC provider"

```

The `createJDBCProvider()` method accepts the following parameters:

- scope - specifies the scope of the JDBC provider
- jdbcDBType - the type of database the provider represents
- jdbcProviderType - the type of JDBC data source provider
- jdbcImplementationType - whether this provider is a connection pool or XA data source provider



### Hint

The Information Center page that you looked at while writing the unit test driver gives you information on the format of each of the variables to be defined next. However, it does not go all the way in giving you information about the contents of the `-jdbcProviderType` parameter.

To find out which values are acceptable, turn to the WebSphere administrative console and go through the steps of creating a “test” JDBC provider at the cluster scope, which uses a DB2 database. Note which values are available on the drop-down for the JDBC provider type. For this lab, you want to use the DB2 Universal JDBC Driver Provider. If you continue to the end of the wizard and click **Finish**, you can then view the command that is generated through the console command assist feature.

**Administrative Scripting Commands**

Administrative Scripting Commands

The wsadmin scripting commands that map to actions on the administrative console display in the Jython language.

**Preferences**

**Administrative Scripting Command**

```
AdminTask.createJDBCProvider(['-scope Cluster=PlantsCluster -databaseType DB2 -providerType "DB2 Universal JDBC Driver Provider" -implementationType "XA data source" -name "DB2 Universal JDBC Driver Provider (XA)" -description "Two-phase commit DB2 JCC provider that supports JDBC 3.0. Data sources that use this provider support the use of XA to perform 2-phase commit processing. Use of driver type 2 on the application server for z/OS is not supported for data sources created under this provider." -classpath ['$DB2UNIVERSAL_JDBC_DRIVER_PATH/db2jcc.jar ${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar'] -nativePath ['$DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH']] )'
```

# Note that scripting list commands may generate more information than is displayed by the administrative console because the console generally filters with respect to scope, templates, and built-in entries.

```
AdminConfig.list('JDBCProvider', AdminConfig.getid('/Cell:was85hostCell01/ServerCluster:PlantsCluster'))
```

Total 3

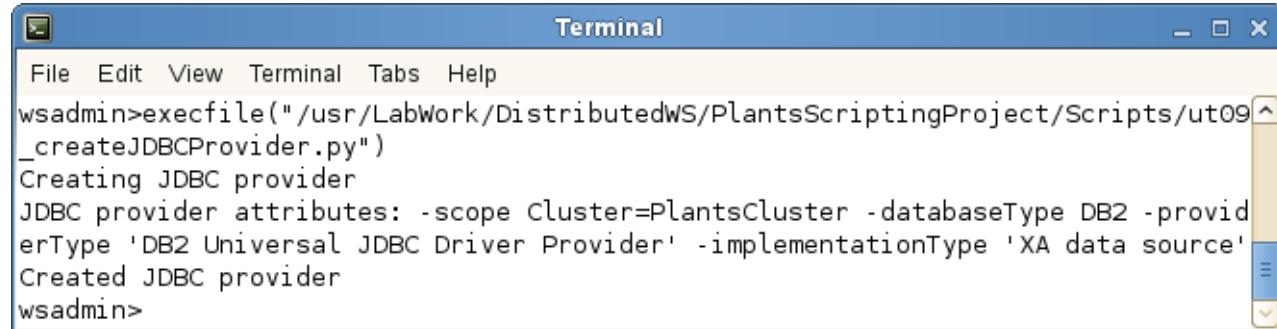
Do not click Save. Log out of the administrative console, and then click **Discard**, then **Yes**.

- \_\_\_ c. Initialize the `scope`, `dbType`, `providerType`, and `implementationType` variables.
- \_\_\_ d. Initialize the `jdbcAttrs` variable by concatenating the variables in the previous step.

- \_\_\_ e. Use the `createJDBCProvider()` method of the AdminTask object to create a JDBC provider.
  - \_\_\_ f. Use AdminConfig to save the configuration changes. Comment out this line.
  - \_\_\_ g. Save the changes to the script.
- \_\_\_ 3. Run `ut08_createJDBCProvider.py` from the wsadmin shell.

- \_\_\_ a. Enter the following command:

```
execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject
/Scripts/ut09_createJDBCProvider.py")
```



The screenshot shows a terminal window with the title "Terminal". The menu bar includes "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area displays the following command and its output:

```
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut09
_createJDBCProvider.py")
Creating JDBC provider
JDBC provider attributes: -scope Cluster=PlantsCluster -databaseType DB2 -prov
iderType 'DB2 Universal JDBC Driver Provider' -implementationType 'XA data source'
Created JDBC provider
wsadmin>
```



### Warning

Running these script multiple times can create multiple JDBC providers. The extra providers are not desirable. In a production script, you have to take care of the situation where the JDBC provider exists and create one only if it does not exist.

During script development, a common practice is to comment out the `AdminConfig.save()` line until all the bugs are fixed; then after all errors are fixed, run the script one last time to save the configuration.

For an extra challenge, you can add logic to the script to create the JDBC provider only if it does not exist.

- \_\_\_ b. Examine the output from this command in the terminal window. Correct any errors that might show up and rerun until successful.
  - \_\_\_ c. Uncomment the `AdminConfig.save()` line in the script and run one more time to create and save the JDBC provider.
- \_\_\_ 4. Verify the JDBC provider was created.
- \_\_\_ a. Log on to the WebSphere administrative console.
  - \_\_\_ b. From the navigation pane, select **Resources > JDBC > JDBC providers**

\_\_\_ c. Select Cluster=PlantsCluster

The screenshot shows the 'JDBC providers' page in the WebSphere administrative console. At the top, there is a scope selection dropdown set to 'Cluster=PlantsCluster'. Below it, a note explains that the scope specifies the level at which the resource definition is visible. The main table lists one provider:

Select	Name	Scope	Description
<input type="checkbox"/>	<a href="#">DB2 Universal JDBC Driver Provider (XA)</a>	Cluster=PlantsCluster	Two-phase commit DB2 JCC provider that supports JDBC 3.0. Data sources that use this provider support the use of XA to perform 2-phase commit processing. Use of driver type 2 on the application server for z/OS is not supported for data sources created under this provider.

Total 1

The JDBC Provider was successfully created at the cluster scope level.

\_\_\_ d. Log out of the administrative console.

## Create datasources

The PlantsByWebSphere application uses a datasource to gain access to the database tables that contain the data that the application needs.

In this part, you create a script capable of creating datasources that a JDBC provider. The data source is named Plants and the PlantsByWebSphere application uses it to store its data.

Your specific requirements are as follows:

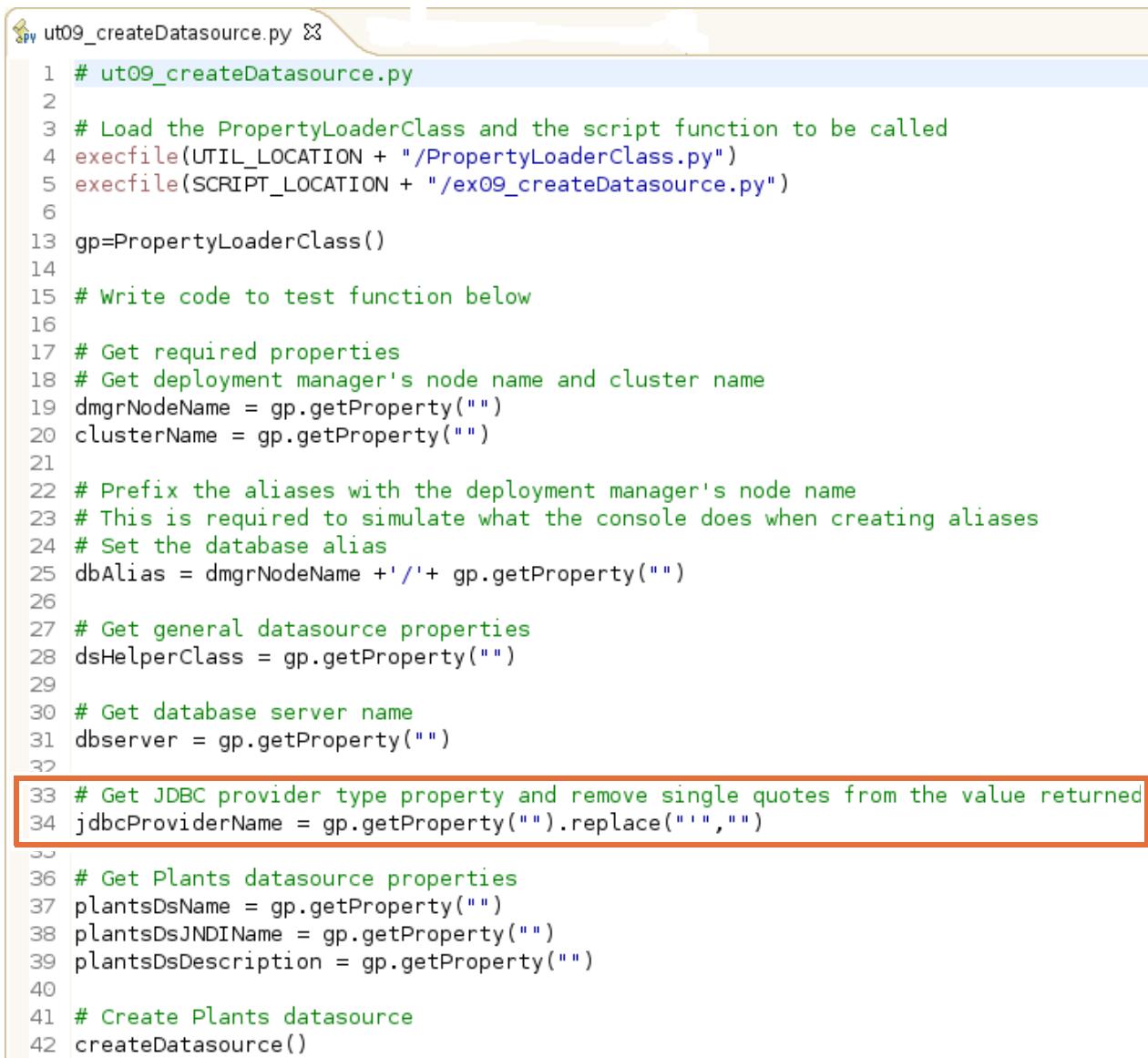
1. Create a Python function named `createDatasource` in a file that is called `ex09_createDatasource.py` to create datasources. The following parameters are passed in to the function:
  - `clusterName` - name of the cluster where the JDBC provider is defined
  - `dbAlias` - an authentication alias that is valid in the database server

- `dsName` - the name of the data source
- `dsJNDIName` - the JNDI name of the data source (must be unique)
- `dsDescription` - a description of the data source
- `dsHelperClass` - the package name of the data store helper class
- `jdbcProviderName` - the name of the JDBC provider that services this data source
- `dbserver` - the host name or IP address of the database server

2. Create a unit test driver, `ut09_createDatasource.py`, which sets up the necessary constants and parameters to invoke the `createDatasource()` function and create the data source required: Plants.

\_\_\_ 1. Edit the `ut06.createDatasource.py` unit test script.

\_\_\_ \_\_\_ a. Review the `ut09_createDatasource.py` unit test script in the Jython editor.



```
1 # ut09_createDatasource.py
2
3 # Load the PropertyLoaderClass and the script function to be called
4 execfile(UTIL_LOCATION + "/PropertyLoaderClass.py")
5 execfile(SCRIPT_LOCATION + "/ex09_createDatasource.py")
6
13 gp=PropertyLoaderClass()
14
15 # Write code to test function below
16
17 # Get required properties
18 # Get deployment manager's node name and cluster name
19 dmgrNodeName = gp.getProperty("")
20 clusterName = gp.getProperty("")
21
22 # Prefix the aliases with the deployment manager's node name
23 # This is required to simulate what the console does when creating aliases
24 # Set the database alias
25 dbAlias = dmgrNodeName +'/' + gp.getProperty("")
26
27 # Get general datasource properties
28 dsHelperClass = gp.getProperty("")
29
30 # Get database server name
31 dbserver = gp.getProperty("")
32
33 # Get JDBC provider type property and remove single quotes from the value returned
34 jdbcProviderName = gp.getProperty("").replace("'", "")
35
36 # Get Plants datasource properties
37 plantsDsName = gp.getProperty("")
38 plantsDsJNDIName = gp.getProperty("")
39 plantsDsDescription = gp.getProperty("")
40
41 # Create Plants datasource
42 createDatasource()
```

- \_\_\_ b. In the appropriate lines of code, retrieve the required constants from the properties file by using the **gp** instance of the property loader class. Here is a list of the required properties that exist in the `plants.properties` file:

- DMGR\_NODE\_NAME
- CLUSTER\_NAME
- AUTH\_ALIAS\_NAME
- DATASOURCE\_HELPER\_CLASSNAME
- DB\_SERVER
- JDBC\_PROVIDER\_NAME
- PLANTS\_DATASOURCE\_NAME
- PLANTS\_DATASOURCE\_JNDI\_NAME
- PLANTS\_DATASOURCE\_DESCRIPTION

- \_\_\_ c. By now you should be able to code the unit test script with little, or no difficulty.



### Information

Constants that contain spaces are enclosed in single quotes in the `plants.properties` file. The use of single quotes makes them easier to use in the scripts because you do not have to use string concatenation to add the quotes before you can use the variables.

However, the single quotes can get in the way when the constants are used in comparisons, either in your code or by code in library functions. In this case, the single quotes must be removed before using the constant.

A simple way to remove the single quotes is to use the **replace()** method of the string object as shown on line 34 of `ut09_createDatasource.py`.

```
jdbcProviderName = gp.getProperty("JDBC_PROVIDER_NAME").replace("'", "")
```

Getting the JDBC provider name from the properties file is the same as you did many times in the past. Once you have the string that represents the provider, you invoke the **replace()** method of the string class, this method looks for the first character that is passed in and replaces it with the second parameter. In this case, the second parameter is an empty string, which results in replacing the single quote with nothing, in effect it removes all single quotes from the string.

Review the code on line 34 and make sure that you understand how single quotes are being removed.

- \_\_\_ d. Complete the code to call the **createDatasource()** method with the right number of parameters, in the right order.

**Hint**

See method signature on the ex09\_createDatasource.py script

- \_\_ e. Save your changes. You test the script later after the script that contains the function is completed.
- \_\_ 2. Edit the ex09\_createDatasource.py script.
  - \_\_ a. Open the **ex09\_createDatasource.py** script.
  - \_\_ b. Review the partially completed **createDatasource()** method.

The **createDatasource()** method accepts the following parameters:

- `clusterName` - specifies the name of the cluster where the JDBC provider is defined
- `jdbcProviderName` - the name of the JDBC provider that services the data source
- `dbServer` - the host name or IP address of the database server
- `dbAlias` - the JAAS alias to be used for database access
- `dsName` - the data source name
- `dsJNDIName` - the JNDI name for the data source
- `dsDescription` - the description for the data source
- `dsHelperClass` - the package name of the data store helper class

Next, the parameters to the `createDatasource()` method are composed. Finally, before the call to create the data source, the individual parameters are combined into a list.

The actual call to create the data source is enclosed in a **try/except** block to catch any exceptions that might occur and to provide an appropriate message whether the function succeeds or fails.

If the creation of the data source completes the configuration is save, otherwise reset.



## Information

If you want to see what the exception text is in order to troubleshoot the problem, you might want to add the following lines to the exception handling block:

```
print "An exception has occurred while running the script."
print "Exception type:", sys.exc_info() [0]
print "Exception value:", sys.exc_info() [1]
print "The filename, line number, function, statement and value of the
exception are:"
print traceback.extract_tb(sys.exc_info() [2])
```

In fact, making a utility function to print the exception information in case of an exception is a good idea. You could create a function that is called ***displayException()*** and call it from within the **except** block in all your functions.

You would put this, and other utility functions in a script that is loaded when wsadmin starts, for example a profile that is loaded at startup before any scripts are run.

\_\_\_ c. Complete the missing code to initialize the following variables:

- dsAttrs
- jndiAttrs
- dbHelperClass
- dbAuthAlias
- dbXARecAlias
- dbResProps



## Hint

To find out the format of the variable, consult the Information Center by searching for: AdminTask createDatasource. As you might remember from the lecture, JDBCProviderManagement is one of the command groups for the AdminTask object. This page on the Information Center can also help you compose the command to actually create the data source.

If you need more help, create the data source by using the WebSphere administrative console and review the results of the console assist page. Do not save the data source definition that created within the console. You might need to use the command assist feature to figure out how to properly format the **dbResProps** variable. The page on the Information Center hints as to how to do it under **configureResourceProperties**, but does not give an example.

The `-configureResourceProperties` parameter is a list of lists. Each individual list describes the name, the Java type, and the value of the parameter. For creating the data source you need four lists within the list:

```
-databaseName, java.lang.String, dsName
-driverType, java.lang.Integer, 4
-serverName, java.lang.String, dbServer
-portNumber, java.lang.Integer, 50001
```

- \_\_ d. Combine all the variables into a list and store in the `createDsParams` variable. Although this step is not necessary, it improves the readability of the script.
- \_\_ e. Use `AdminConfig.getId()` to get the configuration ID of the data source's parent, the JDBC provider. Getting the syntax right might become a bit tricky since you need both the cluster name and the name of the JDBC provider.



### Hint

To find out which object type the cluster is, try using `print AdminConfig.types()`, and look through the existing types to figure out the type names for the cluster and the JDBC provider.

- \_\_ f. Finally, invoke `AdminTask.createDatasource()` passing the `jdbcProviderId` and the `createDsParams` variables.
- \_\_ g. If no exceptions occurred, use `AdminConfig` to save the configuration changes. Comment out this line after all syntax and runtime errors are resolved.
- \_\_ h. If exception did occur, reset the configuration and print an appropriate message.
- \_\_ i. Save the changes to the script.

- \_\_ 3. Run `ut09_createDatasource.py` from the wsadmin shell.

- \_\_ a. Enter the following command:

```
execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject
/Scripts/ut09_createDatasource.py")
```

```
Terminal
File Edit View Terminal Tabs Help
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut09_
_createDatasource.py")
Creating datasource: Plants
Created datasource: Plants
wsadmin>
```

- \_\_\_ b. Examine the output from this command in the terminal window. Correct any errors that might show up and rerun until successful.
  - \_\_\_ c. Uncomment the `AdminConfig.save()` line in the script and run one more time to create and save the datasources.
- \_\_\_ 4. Verify the datasources now exist.
- \_\_\_ a. Log on to the WebSphere administrative console.
  - \_\_\_ b. Select **Resources > JDBC > Data sources**.
  - \_\_\_ c. Select the scope **Cluster=PlantsCluster**.

Name	JNDI name	Scope	Provider	Description	Category
Plants	jdbc/PlantsByWebSphereDataSource	Cluster=PlantsCluster	DB2 Universal JDBC Driver Provider (XA)	Plants by WebSphere application datasource	

- \_\_\_ d. Click the Plants data source and verify that it was created according to the parameters used.
- \_\_\_ e. If the node agents are running, you might test the data sources to make sure that a connection can be made to the database. If you click the **Test Connection** button, you see the following messages..

**Messages**

- [green info icon] The test connection operation for data source Plants on server nodeagent at node PlantsNode01 was successful.
- [green info icon] The test connection operation for data source Plants on server nodeagent at node PlantsNode02 was successful.



## Troubleshooting

---

### **Data source connection failure: Null userid is not supported.**

The data source connection might fail with a message such as: Null userid is not supported. If you see this problem, try restarting the node agents.

---

- f. If you started the node agents to test the connection, stop them now before proceeding to the next section.
- g. Log out of the administrative console.

## Section 3: Install the PlantsByWebSphere application

Now that the resources for the PlantsByWebSphere application are configured you are ready to install the applications themselves.

In addition to installing the **PlantsByWebSphere** application, you also map the application modules to the cluster and web server.

- 1. Make sure that the following EAR file for the application is in the `<profile_root>/Dmgr/installableApps` folder:

- PlantsByWebSphere.ear

If the file is not there, copy it from the `/usr/Software/Ears` folder.

- 2. Open the **ex09\_installPlantsToCluster.py** script. This script is already completed, and in the following steps you review the existing code.
- a. There are three methods that are associated with installing applications.

```

3@def checkIfAppExists(appName):
4    # Returns -1 if application does not exist
5    installedApps = AdminApp.list()
6    return installedApps.find(appName)

8@def unInstallApplication(appName):
9    print "Uninstalling application:", appName
10   AdminApp.uninstall(appName)
11   #Save configuration
12   AdminConfig.save()

14@def installApplicationToCluster(appName, earLocation, clusterName):
15    try:
16        print "Installing application", appName
17        earFileName=appName + ".ear"
18        fqAppName = earLocation + "/" + earFileName
19
20        appOptions = ['-verbose -distributeApp -cluster', clusterName]
21
22        AdminApp.install(fqAppName, appOptions)
23        AdminConfig.save()
24        print "Installed application", appName
25    except:
26        print "Did not Install application", appName
27        print "possibly because it already exists"
28
29
30

```

The method that actually installs the applications is the third one in the group that is shown, `installApplicationToCluster()`. It takes the application's name, the location of the ear file and the cluster name as parameters.

**Note**

This method assumes you are installing to a cluster. Installing an application to a stand-alone server is similar. The only exception is that instead of passing the cluster name by using `-cluster`, you pass the server and node names and use `-server` and `-node` instead.

The installation is enclosed in a **try/except** block to handle the exception that occurs if the application exists. However, as you are going to see in the unit test script, that exception is handled there in a more proactive way.

The first method, `checkIfApplicationExists()` receives the application's name as a parameter and returns a value of `-1` if the application does not exist, any other value indicates the application is already installed. The unit test script calls this method.

If the application exists, it is uninstalled before proceeding. The function that performs the uninstall is `unInstallApplication()`. It receives the application's name as the only parameter.

- \_\_ b. When installing an application, its modules need to be mapped to the available clusters and servers. For the PlantsByWebSphere application, all modules are mapped to the cluster. In addition, the web modules must also be mapped to the web server. This requirement is new in the later versions of WebSphere Application Server and allows the application server to generate correct `plugin-config.xml` files for the web server's plugin.

The `mapModulesToServer()` function accomplishes this task

```

28
29 def mapModuleToServer(app, module, server):
30
31     # Make up complete war file name
32     warName = module + ".war, WEB-INF/web.xml"
33
34     print "Mapping module: " + module + " to server" + server
35
36     AdminApp.edit(app, ["-MapModulesToServers", [[module, warName, server]
37
38     AdminConfig.save()
39

```

The function receives the application's name, the module's name and the server to which the module is mapped. The format of the server is different depending on whether it is single server, a web server, or a cluster of servers. In the next section, you look at the unit test driver to find out about the details.

**Hint**

The details for AdminApp found in the documentation, do not contain enough detail in one place to figure things out. You really need to try the online help through AdminApp.help() in wsadmin, the Information Center, and also the command assist through executing the action through the console and looking at the administrative console's command assistance output. Then, use all the information available to come up with what you need to do.

- \_\_\_ 3. Open the **ut09\_installPlantsToCluster.py** script. This unit test driver script is already completed. In the following steps, you review the existing code.
  - \_\_\_ a. The first part of the script is similar to most other unit test scripts you saw in this exercise..

```

1 # ut09_installPlantsToCluster.py
2
3 # Load the PropertyLoaderClass and the script function to be called
4 execfile(UTIL_LOCATION + "/PropertyLoaderClass.py")
5 execfile(SCRIPT_LOCATION + "/ex09_installTradeToCluster.py")
6
7 # Instantiate property loader
8 gp=PropertyLoaderClass()
9
10 clusterName = gp.getProperty("CLUSTER_NAME")
11 earLocation = gp.getProperty("EAR_LOCATION").replace("'", "")
12 webServerNode = gp.getProperty("IHS_NODE_NAME")
13 webServerName = gp.getProperty("WEB_SERVER_NAME")
14

```

It loads the necessary modules by using the ***execfile()* function**, instantiates the property loader and retrieves the required constants from the properties file. The **earLocation** variable has the single quotes that surround the string constant, in the properties file, removed.

- \_\_\_ b. Now the script gets a little more interesting. Consider the details in the next few steps.

```

15 # Specify the applications names and application modules in a Map
16 appModuleMap={"PlantsByWebSphere": ["PlantsByWebSphereWeb"]}
17
18 # Get keys from dictionary
19 apps = appModuleMap.keys()
20

```

The variable **appModuleMap** contains a Python map. A map, as you recall, is an object that can hold a name/value pair. It acts like a dictionary. You look up the key, or name, and you get the value in return. This map contains the names of

the applications to be installed, these names are the keys of the map. The values for each key represent the web modules in each application. If more than one module exists per application, the modules are represented in a list. If an application has no web modules, the value of **none** is used.

The next step initializes the variable **apps** with the keys from the map. In this case, the keys are the names of the applications.

- c. The next few lines complete the installation and mapping of the modules to servers. The figure illustrates the **for loop** iterating through the applications that are contained in the **apps** variable. For each application, a series of actions takes place.

```

23 # Iterate through the keys
24 for appName in apps:
25
26     # Check if application exists, if it does uninstall it
27     if (checkIfAppExists(appName) != -1):
28         unInstallApplication(appName)
29
30     # Install application
31     installApplicationToCluster(appName, earLocation, clusterName)
32
33     # Get name of cell
34     cellName = AdminControl.getCell()
35
36     # Map modules in application to servers
37     for module in appModuleMap[appName]:
38         if(module <> "none"):
39             # Map module to Web server
40             webServer = "+WebSphere:cell=" + cellName + \
41                         ",node=" + webServerNode + \
42                         ",server=" + webServerName
43
44             → mapModuleToServer(appName, module, webServer)
45
46             # Map module to cluster
47             cluster = "+WebSphere:cell=" + cellName + \
48                         ",cluster=" + clusterName
49
50             → mapModuleToServer(appName, module, cluster)
51

```

Section 1 checks to see whether the application to be installed exists. If a value different from -1 is returned from the `checkIfAppExists()` function, the application is already installed. In this case, the appropriate action is to remove, or uninstall, the application.

Now that you are sure that the application is not installed, the code in section 2 installs the application to the cluster by calling the

`installApplicationToCluster()` function. Note the parameters that are passed.

The name of the cell is required to map the modules to servers. The cell name is obtained at run time, on line 34, using `AdminControl.getCell()`.

Section 3 shows another **for** loop, this time it iterates through all the modules for each application. Notice how the list of modules is retrieved by using the `appName` as the key into the map. If the module is equal to **none**, nothing is done. If there is a module name, the following actions occur:

- i. A string that represents the fully qualified name of the web server is composed. The correct string's format contains the cell, node, and server name in the format that is shown here:

```
+WebSphere:cell=was85hostCell01,node=ihsnode,server=webserver1
```

- ii. Line 44 calls the `mapModulesToServers()` function to map the Web Module to the web server.
- iii. Line 47 makes up a string with the fully qualifies cluster name in the format that is shown here:

```
+WebSphere:cell=was85hostCell01,cluster=PlantsCluster
```

- iv. Since web modules need to be mapped to the cluster (or stand-alone servers), as well as the web server, `mapModulesToServers()` is called again.



### Information

The `mapModulesToServers()` function can be called multiple times to map a module to as many servers (or clusters) as necessary. The **plus** + sign at the beginning of the server name indicates that the mapping is going to be added to any existing mappings. You can also remove a mapping by using a **minus** - sign in front of the server name instead.

— 4. Run `ut09_installPlantsToCluster.py` from the wsadmin shell.

— a. Enter the following command:

```
execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject
/Scripts/ut09_installPlantsToCluster.py")
```

- \_\_\_ b. If the installation and module mapping is successful, you see messages like the following.

```
wsadmin>execfile("/usr/LabWork/DistributedWS/PlantsScriptingProject/Scripts/ut09_installPlantsToCluster.py")
Installing application PlantsByWebSphere
ADMA5016I: Installation of PlantsByWebSphere started.
ADMA5058I: Application and module versions are validated with versions of deployment targets.
ADMA5005I: The application PlantsByWebSphere is configured in the WebSphere Application Server repository.
ADMA5081I: The bootstrap address for client module is configured in the WebSphere Application Server repository.
ADMA5053I: The library references for the installed optional package are created.
ADMA5005I: The application PlantsByWebSphere is configured in the WebSphere Application Server repository.
ADMA5001I: The application binaries are saved in /opt/IBM/WebSphere/AppServer/profiles
/Dmgr/wstemp/Script13cfb04dbde/workspace/cells/was85hostCell01/applications/PlantsByWebSphere.ear/PlantsByWebSphere.eai
ADMA5005I: The application PlantsByWebSphere is configured in the WebSphere Application Server repository.
SECJ0400I: Successfully updated the application PlantsByWebSphere with the appContextIDForSecurity information.
ADMA5113I: Activation plan created successfully.
ADMA5011I: The cleanup of the temp directory for application PlantsByWebSphere is complete.
ADMA5013I: Application PlantsByWebSphere installed successfully.
Installed application PlantsByWebSphere
```

```
Mapping module: PlantsByWebSphereWeb to server+WebSphere:cell=was85hostCell01,node=ihsnode,server=webserver1
ADMA5075I: Editing of application PlantsByWebSphere started.
ADMA5058I: Application and module versions are validated with versions of deployment targets.
<....>
ADMA5113I: Activation plan created successfully.
ADMA5011I: The cleanup of the temp directory for application PlantsByWebSphere is complete.
ADMA5076I: Application PlantsByWebSphere edited successfully.
The application or its web modules may require a restart when a save is performed.
```

```
Mapping module: PlantsByWebSphereWeb to server+WebSphere:cell=was85hostCell01,cluster=PlantsCluster
ADMA5075I: Editing of application PlantsByWebSphere started.
ADMA5058I: Application and module versions are validated with versions of deployment targets.
<....>
ADMA5113I: Activation plan created successfully.
ADMA5011I: The cleanup of the temp directory for application PlantsByWebSphere is complete.
ADMA5076I: Application PlantsByWebSphere edited successfully.
The application or its web modules may require a restart when a save is performed.
```

- \_\_\_ c. Correct any errors that might show up and rerun until successful.
- \_\_\_ 5. If the scripts ran without failure, use the WebSphere administrative console to verify that the application was installed and the module was mapped correctly.
- \_\_\_ a. Log in to the WebSphere administrative console and select **Applications** >**Application Types** > **WebSphere enterprise applications**.

Select	Name	Application Status
<input type="checkbox"/>	PlantsByWebSphere	

Total 1

- \_\_\_ b. Verify that the PlantsByWebSphere application is installed. Depending on what state your server is in, there might be other applications that are already installed

on your system. Also, if the node agents are stopped, the status of the PlantsByWebSphere application is shown as unavailable.

- \_\_\_ 6. Verify that the web modules were properly mapped to the web server and cluster. EJB modules need to be mapped only to the cluster as they are not directly accessible through the web server.
- \_\_\_ a. Click **PlantsByWebSphere**. Under **Modules**, click **Manage Modules**.

Select	Module	URI	Module Type	Server
<input type="checkbox"/>	PlantsByWebSphere	PlantsByWebSphereWeb.war,WEB-INF/web.xml	Web Module	WebSphere:cell=was85hostCell01,node=ihsnode,server=webserver1 WebSphere:cell=was85hostCell01,cluster=PlantsCluster

- \_\_\_ b. Verify that the web module is mapped to both the web server and the cluster.
- \_\_\_ c. Log out of the administrative console.

## Section 4: Testing the PlantsByWebSphere application

Now that everything is configured, you get to see the fruit of all your hard work. Test the installed PlantsByWebSphere application to verify that the application server is configured correctly.

- \_\_\_ 1. Start both node agents.
  - \_\_\_ a. From a Terminal window, navigate to  
`<profile_root>/PlantsProfile1/bin`
  - \_\_\_ b. Enter the command: `./startNode.sh`
  - \_\_\_ c. From a Terminal window, navigate to  
`<profile_root>/PlantsProfile2/bin`
  - \_\_\_ d. Enter the command: `./startNode.sh`
- \_\_\_ 2. Start the IHS web server and admin server.
  - \_\_\_ a. From a Terminal window, navigate to `/opt/IBM/HTTPServer/bin`
  - \_\_\_ b. To start the web server enter: `./apachectl start`
  - \_\_\_ c. To start the admin server enter: `./adminctl start`
- \_\_\_ 3. Start the PlantsCluster.
  - \_\_\_ a. Log in to the administrative console and select **Servers > Clusters > WebSphere application server clusters**.
  - \_\_\_ b. Select **PlantsCluster** and click **Start**.
  - \_\_\_ c. Wait for both servers in the cluster to start.

- \_\_\_ 4. Access the PlantsByWebSphere application.
- \_\_\_ a. Open a web browser by using the following web address to access the PlantsByWebSphere application:

**<http://was85host/PlantsByWebSphere>**

- \_\_\_ b. Click the **Help** link.
- \_\_\_ c. Scroll down on the Help page and click **View Server Info**.

- \_\_\_ d. You see which cluster member received the request..

Cell	Node	Process	Session Data	Session Created
was85hostCell01	PlantsNode01	Plants1	null	null

Session Data

Tue Feb 19 14:15:30 EST 2013

Powered by **IBM WebSphere** e-business software

- \_\_\_ 5. Test cluster member failover.

- \_\_\_ a. From the administrative console, stop the cluster member that received the request.
- \_\_\_ b. In the same browser as before, enter the PlantsByWebSphere web address again: <http://was85host/PlantsByWebSphere>

Your shopping cart is currently empty

HOME : SHOPPING CART : LOGIN : HELP :

Gardens of Summer

They all start with the right flowers... and we've got them all

- \_\_\_ c. Click the **Help** link, and then click **View Server Info**. You now see the other cluster member is serving the request.

The screenshot shows a web application titled "Plants Server Information". At the top, there's a navigation bar with links for HOME, ADMIN HOME, and HELP. Below the navigation, a section titled "PLANTS BY WEBSPHERE" displays "Runtime server information". A table lists the following data:

Cell	Node	Process	Session Data	Session Created
was85hostCell01	PlantsNode02	Plants2	null	null

Below the table, there are buttons for Session Data (with a text input field), Update, and Refresh. A "Show cookies" button is also present. At the bottom of the page, the date and time are displayed as "Tue Feb 19 14:29:09 EST 2013".

## Section 5: Cleaning up the environment

- \_\_\_ 1. Terminate the wsadmin shell by entering: `quit`.
- \_\_\_ 2. Close any scripts that are still open in the Jython editor.
- \_\_\_ 3. Exit the IBM Assembly and Deploy Tools by selecting **File > Exit**.
- \_\_\_ 4. Stop PlantsCluster from the administrative console, and log out.
- \_\_\_ 5. Stop the node agent on **PlantsProfile2**.
  - \_\_\_ a. In the terminal window for `<profile_root>/PlantsProfile2/bin` (it should still be opened from when you started the node agent), execute the following command:  
`./stopNode.sh -user wasadmin -password web1sphere`
  - \_\_\_ b. Wait until you see the message “Server nodeagent stop completed”.
- \_\_\_ 6. Stop the node agent on **PlantsProfile1**.
  - \_\_\_ a. In the Command Prompt window, navigate to  
`<profile_root>/PlantsProfile1/bin` and execute the following command:  
`./stopNode.sh -user wasadmin -password web1sphere`
  - \_\_\_ b. Wait until you see the message “Server nodeagent stop completed”.
- \_\_\_ 7. Stop the deployment manager server.
  - \_\_\_ a. From the terminal window, stop the deployment manager by entering the following command:  
`./stopManager.sh -username wasadmin -password web1sphere`

**End of exercise**

## Exercise review and wrap-up

In this exercise, you modified Jython scripts that use the administrative objects to deploy and manage the PlantsByWebSphere application. You also modified scripts to configure the datasource that the PlantsByWebSphere application uses.



# Exercise 10.ws\_ant scripting and configuring the service integration bus

## What this exercise is about

In this exercise, you explore the structure and syntax of Ant build files and learn how to import property files into build files. You then modify an existing Ant build file and create targets to start applications, stop applications, and modify application attributes. You use the ws\_ant utility to run the Ant build file. Finally, you run several Ant tasks to configure a service integration bus environment to support JMS applications.

## What you should be able to do

At the end of this exercise, you should be able to:

- Combine individual Jython scripts into one Ant script with multiple tasks
- Use Ant property files
- Use Ant tasks for building application code
- Use Ant tasks for deployment and server operation
- Use Ant tasks to configure a service integration environment

## Introduction

In this exercise, you begin by exploring the provided Ant build file, the properties file, and Jython scripts. You also review the two targets in the build file and discover the syntax that is used to start the wsadmin utility to run Jython scripts.

Then, you run the ws\_ant utility by using the provided build file and run the *init* and *stopApplications* build targets and verify their execution.

Next, you modify the provided build file by creating three new targets to start wsadmin and run the *ex10\_startApplication.py*, *ex10\_updateApplicationAttribute*, and *ex10\_stopApplication.py* Jython scripts.

Finally, you set up the service integration bus and configure an environment for messaging applications.

## Requirements

To complete this exercise, you must have the WebSphere Application Server Network Deployment V8.5 product installed. You also require the IBM Assembly and Deploy Tools for WebSphere Administration V8.5. You must complete the previous labs in which you install the deployment manager profile, two managed profiles, the IBM HTTP Server and WebSphere plug-in. You must also create the PlantsCluster, two cluster members, and configured the web server in the cell configuration and installed the PlantsByWebSphere application and configured its resources.

# Exercise instructions



## Important

The labs use two variables to define various installation paths. On Linux, the variable definitions are as follows:

**<was\_root>**: /opt/IBM/WebSphere/AppServer

**<profile\_root>**: /opt/IBM/WebSphere/AppServer/profiles

## Section 1: Explore the Java scripts

- \_\_\_ 1. Use a file system browser such as Nautilus, and browse to **/usr/Software/Scripts/Exercise10**. The following files are provided:
  - ex10.properties
  - ex10\_build.xml
  - ex10\_profile.py
  - ex10\_stopApplication.py
  - ex10\_startApplication.py
  - ex10\_updateApplicationAttribute.py
- \_\_\_ 2. Explore the ex10.properties file.
- \_\_\_ a. Open the **ex10.properties** file in a text editor such as gedit.

```
ex10.properties X
# Administration ID and password for use by scripts
adminID=wasadmin
adminPasswd=web1sphere
softwareDir=/usr/Software/Scripts/Exercise10/
# Location to install applications on cluster
earLocation=/opt/IBM/WebSphere/AppServer/profiles/Dmgr/installableApps/
# Port number for the Dmgr's SOAP connector address
soapPort=8879
connType=soap
applicationName=PlantsByWebSphere
clusterName=PlantsCluster
# Application attribute to be changed
attributeName=reloadInterval
attributeValue=5
```

The ws\_ant utility loads the properties that are listed. When an Ant task starts a Jython script, these properties are used as arguments to the Jython scripts.

**Table 7: Jython script values**

	<b>Argument name</b>	<b>Description</b>
1	adminID	WebSphere administrative ID
2	adminPassword	WebSphere administrative password
3	softwareDir	The default software directory where the Jython scripts exist
4	soapPort	WebSphere deployment manager SOAP port
5	applicationName	Installed application
6	earLocation	Directory where the ear files are stored.
7	clusterName	The name of the cluster
8	attributeName	Name of the attribute that is modifiable.
9	attributeValue	The new value to be assigned to attribute.

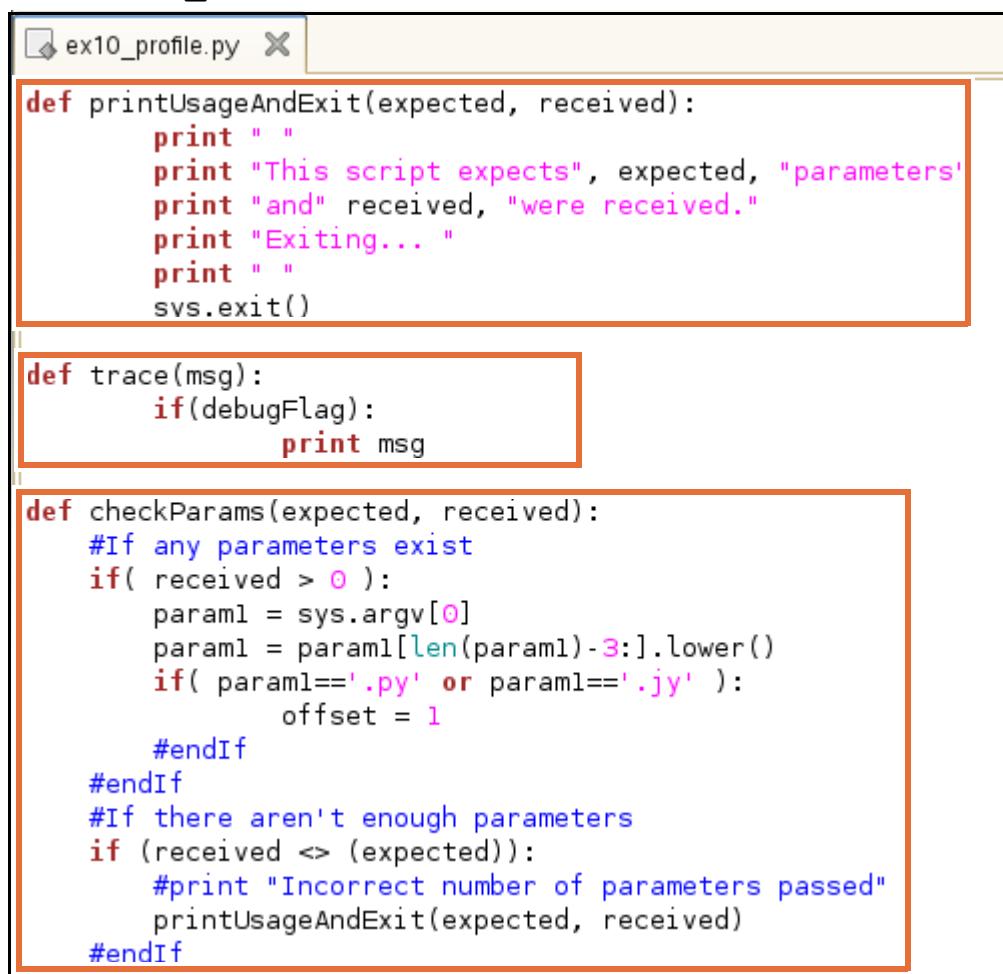
\_\_\_ b. Close the *ex10.properties* file.



### Note

The property names and values are case-sensitive. Single or double quotation marks should not be used, as these characters are passed as part of the variable value. However, quotation marks should be used when defining a list of argument value. For example, the *applicationNames* argument defines a comma delimited list of the installed applications.

3. Explore the ex10\_profile.py Jython script.
- a. Open **ex10\_profile.py** file in a text editor.



```

def printUsageAndExit(expected, received):
    print ""
    print "This script expects", expected, "parameters"
    print "and" received, "were received."
    print "Exiting..."
    print ""
    sys.exit()

def trace(msg):
    if(debugFlag):
        print msg

def checkParams(expected, received):
    #If any parameters exist
    if( received > 0 ):
        param1 = sys.argv[0]
        param1 = param1[len(param1)-3:].lower()
        if( param1=='.py' or param1=='.jy' ):
            offset = 1
        #endif
    #endif
    #If there aren't enough parameters
    if (received <> (expected)):
        #print "Incorrect number of parameters passed"
        printUsageAndExit(expected, received)
    #endif

```

The `ex10_profile.py` Jython script is passed as the profile parameter (`wsadmin.sh -profile ex10_profile.py`) when `wsadmin` is started. The `ex10_profile.py` script defines the `printUsageAndExit()` method that prints a message if an incorrect number of arguments are passed to a Jython script. The `trace()` method prints the specified message if debugging is enabled. The `checkParams()` method ensures that the right number of parameters were sent to the script.

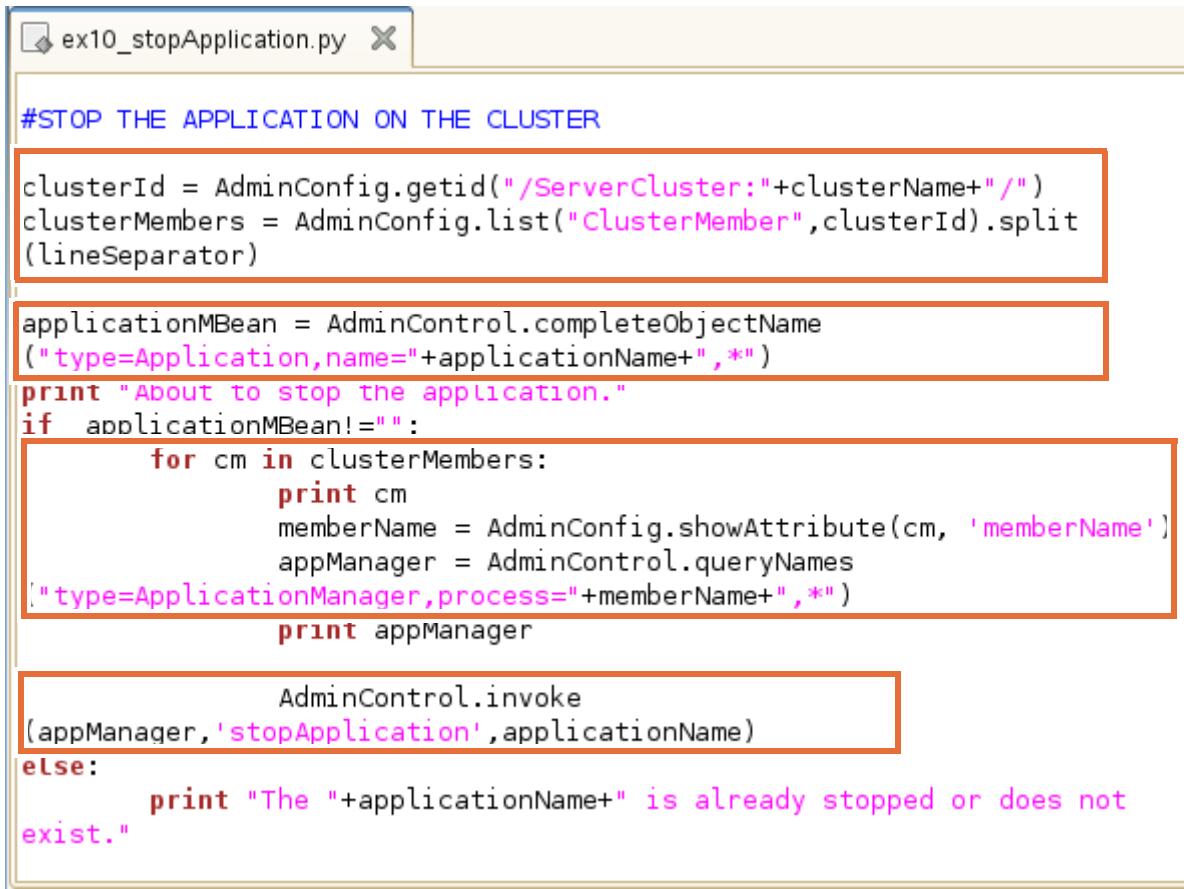
The `dmgrNodeName` variable is assigned to the name of the deployment manager name and the `cell` variable is assigned the deployment manager cell name.



### Information

These methods and variables are available to all Jython scripts run on the `wsadmin` environment that was started by using this profile.

- \_\_ b. Close the `ex10_profile.py` file.
- \_\_ 4. Explore `ex10_stopApplication.py`.
- \_\_ a. Open the `ex10_stopApplication.py` file in a text editor.



```
#STOP THE APPLICATION ON THE CLUSTER

clusterId = AdminConfig.getId("/ServerCluster:"+clusterName+"/")
clusterMembers = AdminConfig.list("ClusterMember",clusterId).split
(lineSeparator)

applicationMBean = AdminControl.completeObjectName
("type=Application,name="+applicationName+",*")
print "About to stop the application."
if applicationMBean!="":
    for cm in clusterMembers:
        print cm
        memberName = AdminConfig.showAttribute(cm, 'memberName')
        appManager = AdminControl.queryNames
(["type=ApplicationManager,process="+memberName+",*"])
        print appManager

        AdminControl.invoke
(appManager,'stopApplication',applicationName)
else:
    print "The "+applicationName+" is already stopped or does not
exist."
```

The script checks if the correct number of arguments are passed. The arguments are assigned to the `applicationName` and `clusterName` variables. (Not shown in the screen capture.)

Next, the `clusterId` and a list of the `clusterMembers` is obtained from the `AdminConfig` administrative object.

A reference to the application's MBean is obtained and assigned to the `applicationMBean` variable. This MBean can be used to manipulate some aspects of the running application. If the application is running the variable has a value, if it is not running the variable has a blank string that is assigned to it.

Since the application is running on multiple servers on the cluster, the `for` loop cycles through the list of cluster members. For each cluster member, the `memberName` is obtained and used to get a reference to the `ApplicationManager` MBean, which is assigned to the `appManager` variable.

Using the `appManager` variable that the application sends to the **stopApplication** command.

Informational messages are printed as required.

- \_\_\_ b. Close the file.
- \_\_\_ 5. Explore the ex10\_startApplication.py.
  - \_\_\_ a. Open **ex10\_startApplication.py** in a text editor.
  - \_\_\_ b. As you can see, starting an application is basically the same as stopping an application. The only difference is the command sent to the *appManager*.
  - \_\_\_ c. Close the file.

## Section 2: Examine the ex10\_build.xml file

In this section, you review the ex10\_build.xml file to understand the syntax of the Ant build files and the Jython scripts that are started.

- \_\_\_ 1. Explore the ex10\_build.xml file.

The ex10\_build.xml file contains the template for building ws\_ant scripts.

- \_\_\_ a. Open **ex10\_build.xml** in a text editor.  
\_\_\_ b. Review the following lines:

```
<taskdef name="wsadmin"  
classname="com.ibm.websphere.ant.tasks.WsAdmin"/>
```

The **<taskdef>** element makes wsadmin available to ws\_ant to run. The **classname** parameter defines the implementation class for the task, in this case: **com.ibm.websphere.ant.tasks.WsAdmin**.

- \_\_\_ c. Notice that the first target defined is named **init**.

```
<target name="init">  
    <!-- Create the time stamp -->  
    <tstamp/>  
    <!-- Read properties from file -->  
    <property resource="ex10.properties"/>  
    <echo message="--- Build of ${ant.project.name} started at  
    ${TSTAMP} on ${TODAY} ---"/>  
    <echo message="--- Using ant version: ${ant.version} --- "/>  
    <echo message="ex10_build.xml"/>  
</target>
```

The **init** target prints the timestamp, the **ant.project.name** property, and the current dates timestamp to the console.

It also loads the common properties for the tasks.



### Information

All the tasks in the build file depend on init. Which means that if another task did not run init previously, init runs before the requested task.

- \_\_\_ d. Review the stopApplication target.

The stopApplication target is shown here.

```
<target name="stopApplication" depends="init"
description="Stops the specified application.">
    <echo message="--- Stopping the ${applicationName}
applications. ---"/>
    <wsadmin script="${softwareDir}ex10_stopApplication.py"
lang="jython" profile="${softwareDir}ex10_profile.py"
port="${soapPort}" conntype="soap" user="${adminID}"
password="${adminPasswd}">
        <arg value="${applicationName}" />
        <arg value="${clusterName}" />
    </wsadmin>
</target>
```

The **stopApplication** target starts the wsadmin script by using a wsadmin Ant task. Review the following wsadmin attributes:

- **script** - specifies the script file to run
- **lang** - wsadmin startup language
- **profile** - the profile script to run on wsadmin invocation
- **port** - WebSphere administrative port
- **conntype** - wsadmin connection type
- **user** - the WebSphere administrative user
- **password** - the WebSphere administrative password

Additionally, the **arg** Ant element defines arguments that are passed to the Jython script specified in the **script** attribute.

The **arg** element accepts the **value** attribute that defines the argument value. To pass values from a properties file, use the  `${propertyName}` construct.



### Important

The element, attributes, and property names are all case sensitive.

2. Review the remaining targets in the build file.

### Section 3: Test the tasks that are defined in the build file

You explored the `ex10_build.xml` file and reviewed the targets that are defined in it. The `init` target outputs basic information to the console. The `stopApplications` target starts `wsadmin` to run the `ex10_stopApplication.py` Jython script. This Jython script iterates through the cluster members and stops an application on all cluster members.

- \_\_\_ 1. Set up the environment by starting the servers and copying the properties file. You use simple shell scripts for this step.
  - \_\_\_ a. Open a terminal window.
  - \_\_\_ b. Start the deployment manager and node agents by running the following script:

```
/usr/Software/Scripts/Exercise10/start_cell.sh
```

```
Terminal
File Edit View Terminal Tabs Help
was85host:~ # /usr/Software/Scripts/Exercise10/start_cell.sh
Starting the cell

Starting Dmgr
Starting profile1
Starting profile2
Wait for messages that show the nodeagents and dmgr are open for e-business
was85host:~ # ADMU0116I: Tool information is being logged in file
               /opt/IBM/WebSphere/AppServer/profiles/PlantsProfile1/logs/nodeagent/s
tartServer.log
ADMU0116I: Tool information is being logged in file
               /opt/IBM/WebSphere/AppServer/profiles/Dmgr/logs/dmgr/startServer.log
ADMU0116I: Tool information is being logged in file
               /opt/IBM/WebSphere/AppServer/profiles/PlantsProfile2/logs/nodeagent/s
tartServer.log
ADMU0128I: Starting tool with the Dmgr profile
ADMU3100I: Reading configuration for server: dmgr
ADMU0128I: Starting tool with the PlantsProfile1 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU0128I: Starting tool with the PlantsProfile2 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server nodeagent open for e-business; process id is 15295
ADMU3000I: Server nodeagent open for e-business; process id is 15302
ADMU3000I: Server dmgr open for e-business; process id is 15290

was85host:~ #
```

- \_\_\_ c. Start the application servers by running the following script:

`/usr/Software/Scripts/Exercise10/start_appservers.sh`

The terminal window shows the command `/usr/Software/Scripts/Exercise10/start_appservers.sh` being run. The output indicates the start of two servers: `Plants1` and `Plants2`. It also includes log entries from the WebSphere Application Server (ADMU) tool, which logs information about the server profiles and their launch. The final message shows both servers are open for e-business.

```

Terminal
File Edit View Terminal Tabs Help
was85host:~ # /usr/Software/Scripts/Exercise10/start_appservers.sh
Starting Plants1
Starting Plants2
Wait for messages showing that servers Plants1 and Plants2 are open for e-business
was85host:~ # ADMU0116I: tool information is being logged in file
          /opt/IBM/WebSphere/AppServer/profiles/PlantsProfile1/logs/Plants1/sta
rtServer.log
ADMU0116I: Tool information is being logged in file
          /opt/IBM/WebSphere/AppServer/profiles/PlantsProfile2/logs/Plants2/sta
rtServer.log
ADMU0128I: Starting tool with the PlantsProfile1 profile
ADMU3100I: Reading configuration for server: Plants1
ADMU0128I: Starting tool with the PlantsProfile2 profile
ADMU3100I: Reading configuration for server: Plants2
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3200I: Server launched. Waiting for initialization status
ADMU3000I: Server Plants2 open for e-business; process id is 17089
ADMU3000I: Server Plants1 open for e-business; process id is 17084

```

- \_\_\_ d. Copy the `ex10.properties` file to the deployment manager's properties directory and `ex10_build.xml` to `<profile_root>/Dmgr/bin` by running the following script:

`/usr/Software/Scripts/Exercise10/copy_props.sh`

The terminal window shows the command `/usr/Software/Scripts/Exercise10/copy_props.sh` being run. The output indicates the properties file is being copied and the process is done.

```

Terminal
File Edit View Terminal Tabs Help
was85host:~ # /usr/Software/Scripts/Exercise10/copy_props.sh
Copying properties file
Done
was85host:~ #

```

- \_\_\_ 2. Run the `ws_ant` utility.
  - \_\_\_ a. Open a terminal window, and go to the `<profile_root>/Dmgr/bin` directory.



## Information

To review ws\_ant usage syntax, run the following command:

```
./ws_ant.sh -help
```

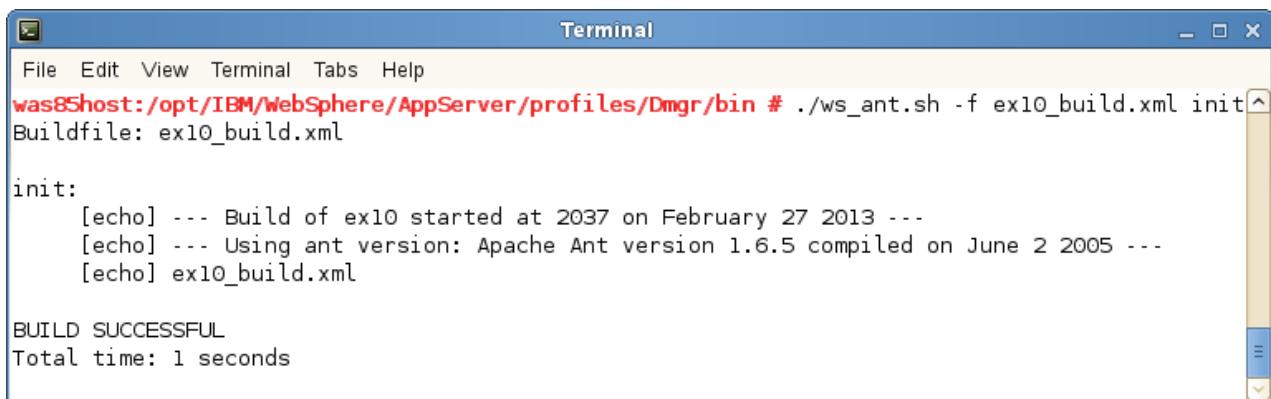
The screenshot shows a terminal window titled "Terminal". The command `./ws_ant.sh -help` is entered, followed by the usage information for the ws\_ant command. The usage information includes options for help, project help, version, diagnostics, quiet mode, verbose mode, debug mode, emacs mode, a lib path, a logfile, buildfiles, properties, keep-going mode, propertyfiles, input handlers, find operations, nice values, no user library, and no classpath. The terminal window has a blue header bar and a white body with black text.

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -help
ant [options] [target [target2 [target3] ...]]
Options:
  -help, -h           print this message
  -projecthelp, -p    print project help information
  -version            print the version information and exit
  -diagnostics        print information that might be helpful to
                      diagnose or report problems.
  -quiet, -q          be extra quiet
  -verbose, -v         be extra verbose
  -debug, -d          print debugging information
  -emacs, -e          produce logging information without adornments
  -lib <path>         specifies a path to search for jars and classes
  -logfile <file>     use given file for log
    -l    <file>          ''
  -logger <classname> the class which is to perform logging
  -listener <classname> add an instance of class as a project listener
  -noinput             do not allow interactive input
  -buildfile <file>   use given buildfile
    -file   <file>          ''
    -f      <file>          ''
  -D<property>=<value> use value for given property
  -keep-going, -k     execute all targets that do not depend
                      on failed target(s)
  -propertyfile <name> load all properties from file with -D
                      properties taking precedence
  -inputhandler <class> the class which will handle input requests
  -find <file>        (s)earch for buildfile towards the root of
    -s  <file>          the filesystem and use it
  -nice  number        A niceness value for the main thread:
                      1 (lowest) to 10 (highest); 5 is the default
  -nouserlib          Run ant without using the jar files from
                      ${user.home}/.ant/lib
  -noclasspath         Run ant without using CLASSPATH
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin #
```

The standard usage syntax is `./ws_ant.sh -f buildfile.xml target_name`

- \_\_ b. Run the following command to run only the *init* target:

```
./ws_ant.sh -f ex10_build.xml init
```



The terminal window shows the command being run and its output. The output includes build information, the XML file used, and a successful build message.

```
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -f ex10_build.xml init
Buildfile: ex10_build.xml

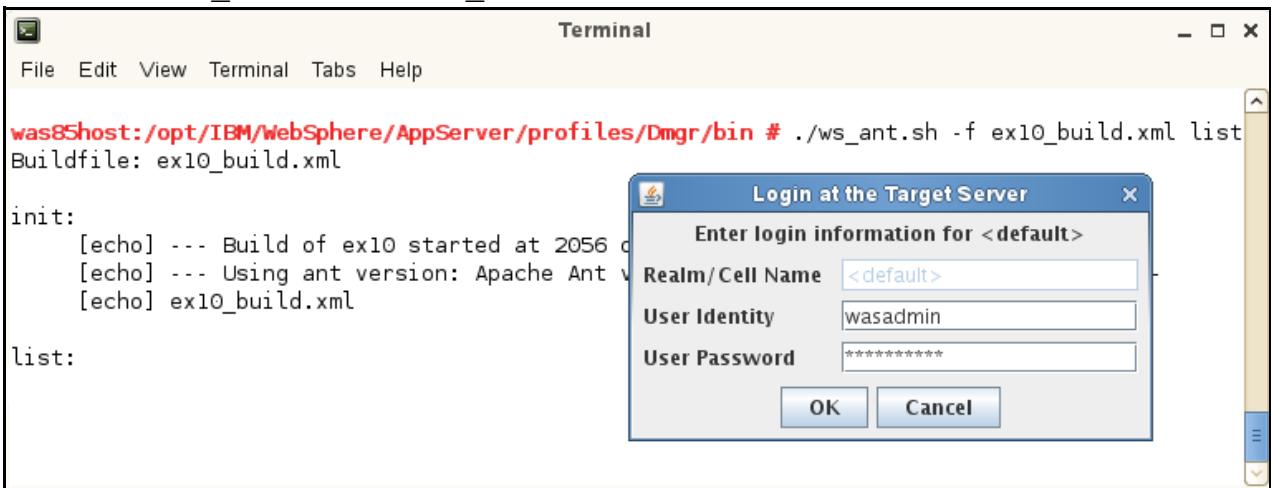
init:
[echo] --- Build of ex10 started at 2037 on February 27 2013 ---
[echo] --- Using ant version: Apache Ant version 1.6.5 compiled on June 2 2005 ---
[echo] ex10_build.xml

BUILD SUCCESSFUL
Total time: 1 seconds
```

The ws\_ant utility is started and the init target is ran. The results are displayed.

- \_\_ c. Run the following command to list the applications that are installed:

```
./ws_ant.sh -f ex10_build.xml list
```



The terminal window shows the command being run. A login dialog box is overlaid on the window, prompting for realm/cell name, user identity, and user password. The user identity is set to 'wasadmin'.

```
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -f ex10_build.xml list
Buildfile: ex10_build.xml

init:
[echo] --- Build of ex10 started at 2056 on February 27 2013 ---
[echo] --- Using ant version: Apache Ant version 1.6.5 compiled on June 2 2005 ---
[echo] ex10_build.xml

list:
```

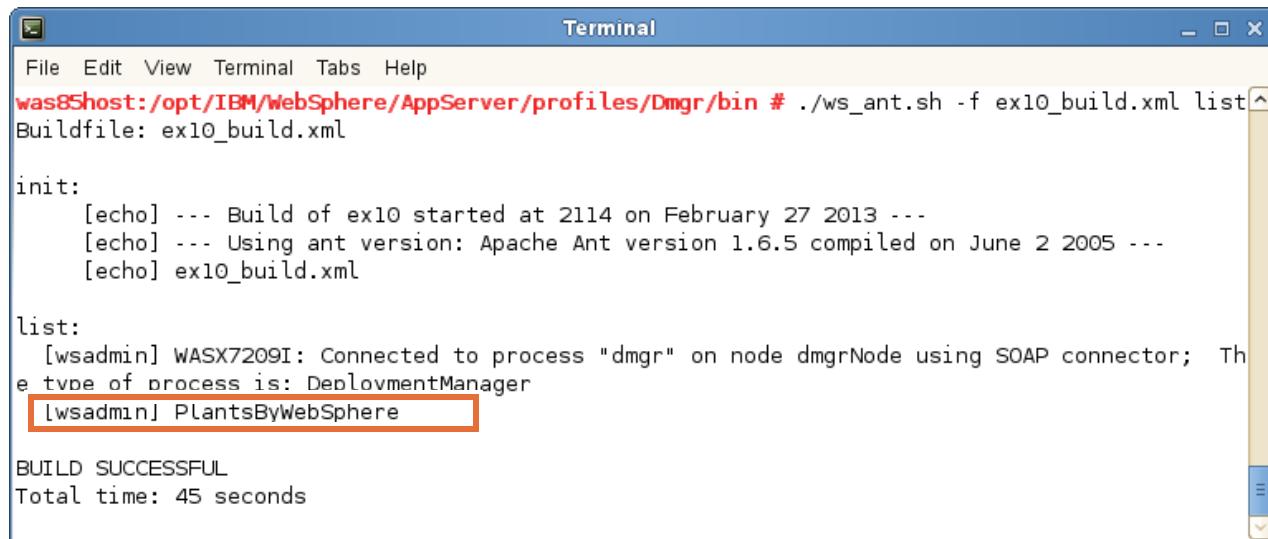
**Login at the Target Server**

Enter login information for <default>

Realm/Cell Name	<default>
User Identity	wasadmin
User Password	*****

OK Cancel

- \_\_\_ d. You are not prompted to log in if you added the user ID and password to the `soap.client.props` file in a previous exercise. Log in at the prompt by using **User ID:** wasadmin and **Password:** web1sphere, and then click **OK**.



```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -f ex10_build.xml list
Buildfile: ex10_build.xml

init:
[echo] --- Build of ex10 started at 2114 on February 27 2013 ---
[echo] --- Using ant version: Apache Ant version 1.6.5 compiled on June 2 2005 ---
[echo] ex10_build.xml

list:
[wsadmin] WASX7209I: Connected to process "dmgr" on node dmgrNode using SOAP connector; The type of process is: DeploymentManager
[wsadmin] PlantsByWebSphere

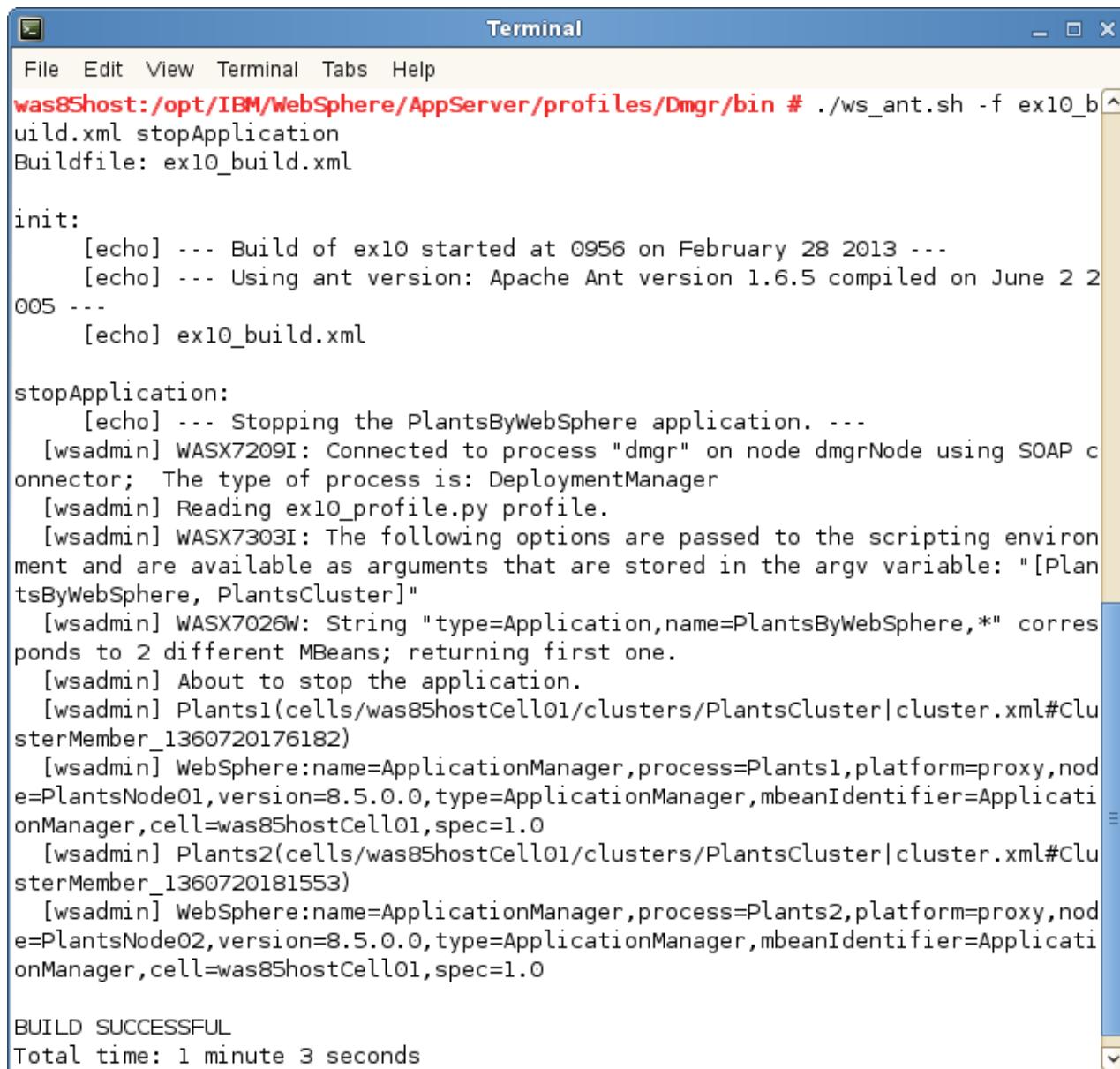
BUILD SUCCESSFUL
Total time: 45 seconds
```

Notice that the target in the `ex10_build.xml` for the `list` command is different than the rest. This is an example of a build in WebSphere `ws_ant` target.

```
<target name="list" depends="init"
       description="Lists all applications">
    <wsListApps/>
</target>
```

- \_\_\_ e. Run the following command to stop the application that is named under the *applicationName* constant that is defined in the properties file (in this lab the PlantsByWebSphere application).

```
./ws_ant.sh -f ex10_build.xml stopApplication
```



The screenshot shows a terminal window titled "Terminal". The command `./ws_ant.sh -f ex10_build.xml stopApplication` is run from the directory `/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin`. The output shows the build process starting at 0956 on February 28, 2013, using Apache Ant version 1.6.5. It connects to the dmgr process on dmgrNode using SOAP and stops the PlantsByWebSphere application. The process involves connecting to two different ApplicationManagers (Plants1 and Plants2) on nodes PlantsNode01 and PlantsNode02 respectively. The application is successfully stopped, and the total build time is 1 minute 3 seconds.

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -f ex10_build.xml stopApplication
Buildfile: ex10_build.xml

init:
[echo] --- Build of ex10 started at 0956 on February 28 2013 ---
[echo] --- Using ant version: Apache Ant version 1.6.5 compiled on June 2 2
005 ---
[echo] ex10_build.xml

stopApplication:
[echo] --- Stopping the PlantsByWebSphere application. ---
[wsadmin] WASX7209I: Connected to process "dmgr" on node dmgrNode using SOAP c
onnector; The type of process is: DeploymentManager
[wsadmin] Reading ex10_profile.py profile.
[wsadmin] WASX7303I: The following options are passed to the scripting environ
ment and are available as arguments that are stored in the argv variable: "[Plan
tsByWebSphere, PlantsCluster]"
[wsadmin] WASX7026W: String "type=Application,name=PlantsByWebSphere,*" corre
sponds to 2 different MBeans; returning first one.
[wsadmin] About to stop the application.
[wsadmin] Plants1(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#Clu
sterMember_1360720176182)
[wsadmin] WebSphere:name=ApplicationManager,process=Plants1,platform=proxy,nod
e=PlantsNode01,version=8.5.0.0,type=ApplicationManager,mbeanIdentifier=Applicati
onManager,cell=was85hostCell01,spec=1.0
[wsadmin] Plants2(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#Clu
sterMember_1360720181553)
[wsadmin] WebSphere:name=ApplicationManager,process=Plants2,platform=proxy,nod
e=PlantsNode02,version=8.5.0.0,type=ApplicationManager,mbeanIdentifier=Applicati
onManager,cell=was85hostCell01,spec=1.0

BUILD SUCCESSFUL
Total time: 1 minute 3 seconds
```

The application that is specified in the *ex10.properties* file (PlantsByWebSphere) is stopped successfully.

- \_\_\_ 3. Verify the application is stopped.
- \_\_\_ a. Log in to the administrative console.
- \_\_\_ b. On the left navigation, click **Applications > Application Types > WebSphere enterprise applications**.

- \_\_\_ c. Verify that the PlantsByWebSphere application is stopped.

The screenshot shows the 'Enterprise Applications' interface. At the top, there is a toolbar with buttons for Start, Stop, Install, Uninstall, Update, Rollout Update, Remove File, Export, and Export File. Below the toolbar is a section with four icons: a checkmark, a square, an upward arrow, and a downward arrow. A search bar has 'Select' and 'Name' dropdowns, and an 'Application Status' button with a circular arrow icon. Below this is a table titled 'You can administer the following resources:' with one row. The row contains a checkbox, the application name 'PlantsByWebSphere', and a red 'X' icon indicating it is stopped. At the bottom left, it says 'Total 1'.



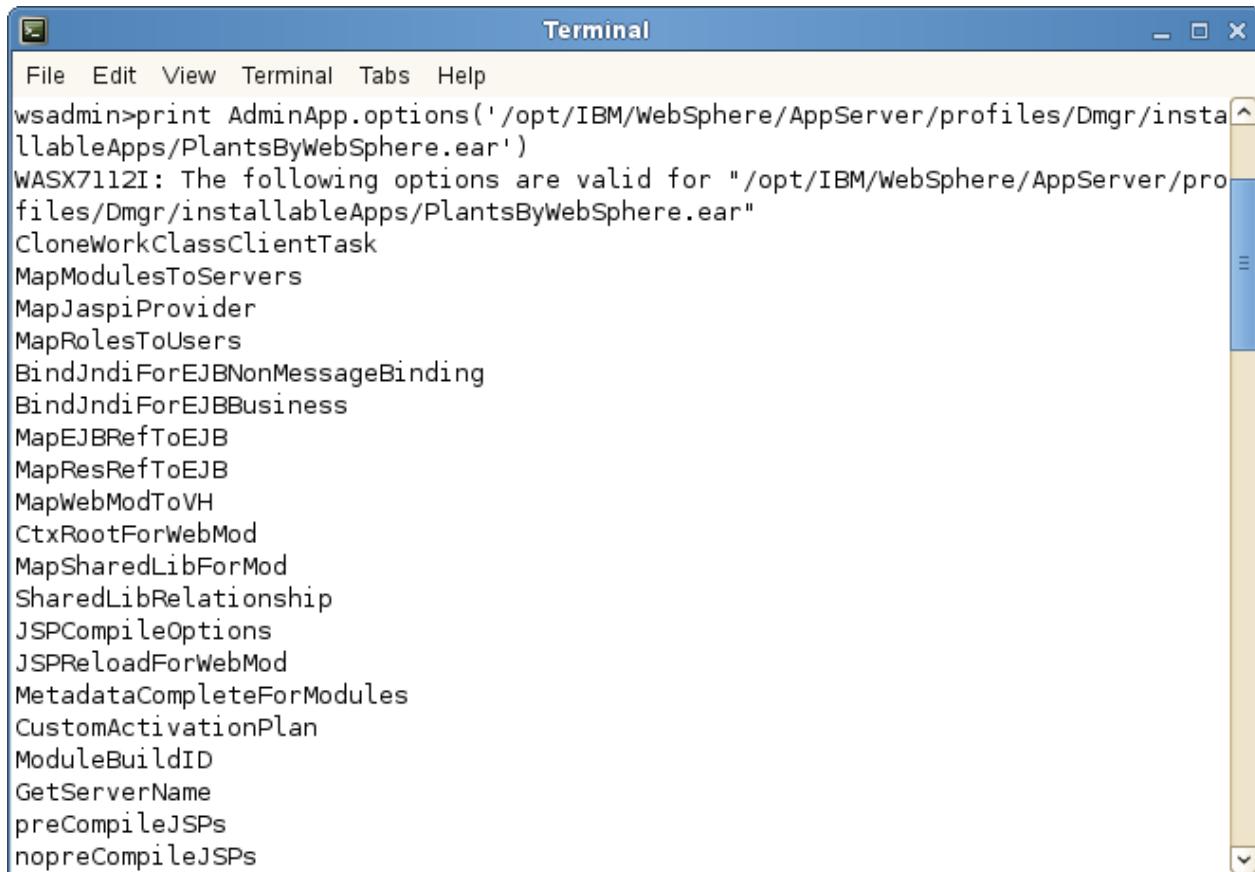
### Information

If you do not see that the application is stopped on the console, you see either a pale green arrow, or a pale red x, indicating a partial start or stop. Check the SystemOut.log files for both servers on the cluster to verify that the application is stopped. It is possible for the console to get out of sync with the actual state of the server.

- \_\_\_ 4. Update an application attribute. An application has many attributes that can be modified by using scripts. To see which attributes can be updated, you can use the AdminApp.options command. For example, to see the modifiable attributes for the PlantsByWebSphere application, you can use the following Jython command.

```
print  
AdminApp.options('/opt/IBM/WebSphere/AppServer/profiles/Dmgr/installableApps/PlantsByWebSphere.ear')
```

Here is a portion of the output from this command.



The screenshot shows a terminal window titled "Terminal". The window contains the following text:

```

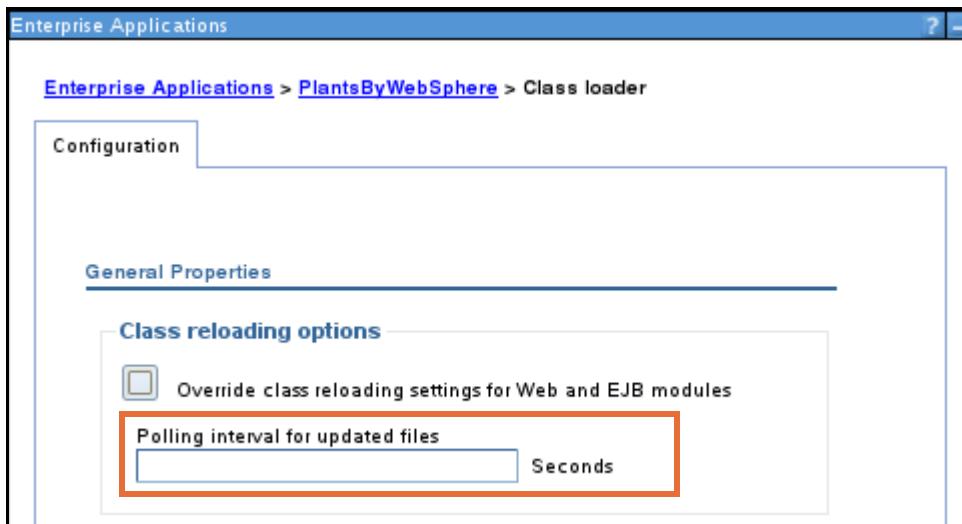
File Edit View Terminal Tabs Help
wsadmin>print AdminApp.options('/opt/IBM/WebSphere/AppServer/profiles/Dmgr/installableApps/PlantsByWebSphere.ear')
WASX7112I: The following options are valid for "/opt/IBM/WebSphere/AppServer/profiles/Dmgr/installableApps/PlantsByWebSphere.ear"
CloneWorkClassClientTask
MapModulesToServers
MapJaspiProvider
MapRolesToUsers
BindJndiForEJBNonMessageBinding
BindJndiForEJBBusiness
MapEJBRefToEJB
MapResRefToEJB
MapWebModToVH
CtxRootForWebMod
MapSharedLibForMod
SharedLibRelationship
JSPCompileOptions
JSPReloadForWebMod
MetadataCompleteForModules
CustomActivationPlan
ModuleBuildID
GetServerName
preCompileJSPs
nopreCompileJSPs

```

Farther down in the list is the attribute that you are going to update, **reloadInterval**. This attribute is the same as the Polling interval for updated files and specifies the number of seconds to scan the application's file system for updated files.

- \_\_\_ 5. Update an application attribute.
  - \_\_\_ a. First, using the console, verify the application attribute's current setting. Go to **Applications > Application Type > WebSphere enterprise applications > PlantsByWebSphere > Class loading and update detection**

- \_\_\_ b. Notice that the **Polling interval for updated files** is blank, though the default setting is 3 seconds.



- \_\_\_ c. Run the following command to update an application attribute. The attribute to be updated (**reloadInterval**), and its new value 5 are defined in the properties file.

```
./ws_ant.sh -f ex10_build.xml updateApplicationAttribute
```

The terminal window title is 'Terminal'. The command run is `./ws_ant.sh -f ex10_build.xml updateApplicationAttribute`. The output shows the buildfile is 'ex10\_build.xml'. The 'init:' section shows the start of the build. The 'updateApplicationAttribute:' section shows the attribute 'reloadInterval' being updated to 5. The message '[wsadmin] WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable: "[PlantsByWebSphere, reloadInterval, 5]"' indicates the attribute was successfully updated. The final message 'BUILD SUCCESSFUL' and 'Total time: 35 seconds' are at the bottom. The entire 'updateApplicationAttribute:' section is highlighted with a red rectangular border.

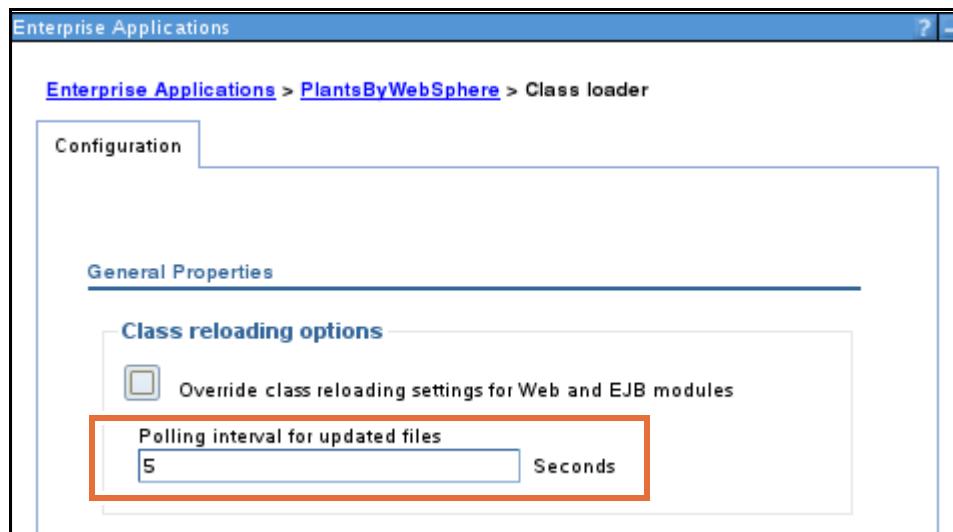
```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -f ex10_build.xml updateApplicationAttribute
Buildfile: ex10_build.xml

init:
[echo] --- Build of ex10 started at 1243 on February 28 2013 ---
[echo] --- Using ant version: Apache Ant version 1.6.5 compiled on June 2 2005 ---
[echo] ex10_build.xml

updateApplicationAttribute:
[echo] --- Updating the PlantsByWebSphere attribute. ---
[wsadmin] WASX7209I: Connected to process "dmgr" on node dmgrNode using SOAP connector; The type of process is: DeploymentManager
[wsadmin] Reading ex10 profile.pv profile.
[wsadmin] WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable: "[PlantsByWebSphere, reloadInterval, 5]"
[wsadmin] The attribute reloadInterval is set to: 3
[wsadmin] The attribute reloadInterval has been changed to the following value: 5

BUILD SUCCESSFUL
Total time: 35 seconds
```

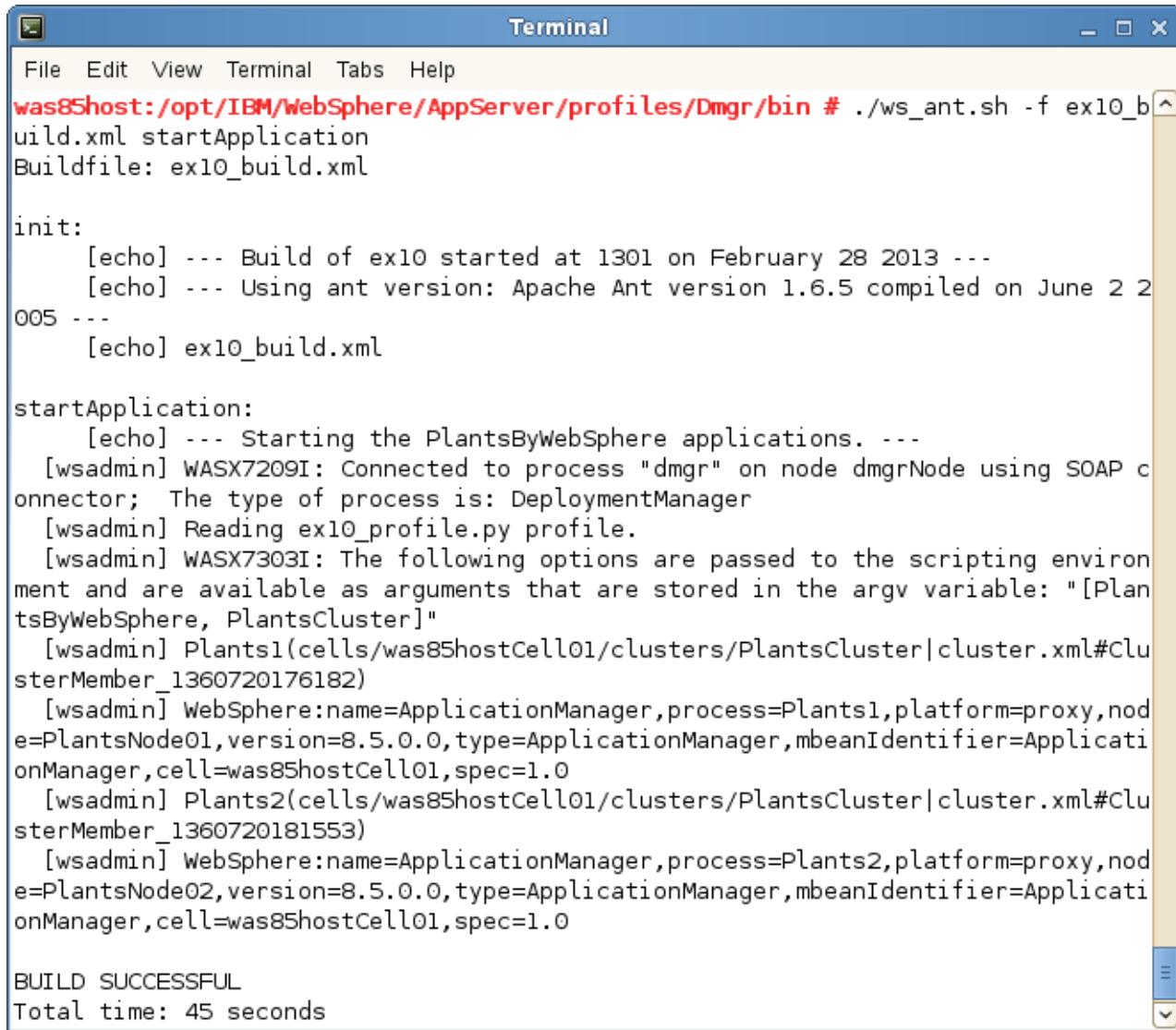
- \_\_\_ d. Return to the console and verify that the setting is changed. You must refresh the page by clicking away and then return to that page once or twice to see the update.



- \_\_\_ 6. Test the *startApplication* target. The application to be started is defined in the properties file, in this case the PlantsByWebSphere.

- \_\_\_ a. Run the following command to start the application by using ws\_ant:

```
./ws_ant.sh -f ex10_build.xml startApplication
```



The screenshot shows a terminal window titled "Terminal". The window contains the output of a shell command. The command is: `./ws_ant.sh -f ex10_build.xml startApplication`. The output shows the build process starting with "init:" and then moving to "startApplication:". It details the connection to dmgr, reading of profiles, and starting of two application processes named Plants1 and Plants2. The build concludes with "BUILD SUCCESSFUL" and a total time of 45 seconds.

```
File Edit View Terminal Tabs Help  
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -f ex10_build.xml startApplication  
Buildfile: ex10_build.xml  
  
init:  
    [echo] --- Build of ex10 started at 1301 on February 28 2013 ---  
    [echo] --- Using ant version: Apache Ant version 1.6.5 compiled on June 2 2005 ---  
    [echo] ex10_build.xml  
  
startApplication:  
    [echo] --- Starting the PlantsByWebSphere applications. ---  
    [wsadmin] WASX7209I: Connected to process "dmgr" on node dmgrNode using SOAP connector; The type of process is: DeploymentManager  
    [wsadmin] Reading ex10_profile.py profile.  
    [wsadmin] WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable: "[PlantsByWebSphere, PlantsCluster]"  
    [wsadmin] Plants1(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#ClusterMember_1360720176182)  
    [wsadmin] WebSphere:name=ApplicationManager,process=Plants1,platform=proxy,node=PlantsNode01,version=8.5.0.0,type=ApplicationManager,mbeanIdentifier=ApplicationManager,cell=was85hostCell01,spec=1.0  
    [wsadmin] Plants2(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#ClusterMember_1360720181553)  
    [wsadmin] WebSphere:name=ApplicationManager,process=Plants2,platform=proxy,node=PlantsNode02,version=8.5.0.0,type=ApplicationManager,mbeanIdentifier=ApplicationManager,cell=was85hostCell01,spec=1.0  
  
BUILD SUCCESSFUL  
Total time: 45 seconds
```

- \_\_\_ b. Use the administrative console to verify that the PlantsByWebSphere application is started. Refresh the application status to show that it is started.

## Section 4: Setting up the service integration bus and configuring an environment for messaging application



### Information

#### Automation and scripting methodology

In this section, the use of ws\_ant is illustrated by creating an environment to support applications that use WebSphere Default Messaging and service integration bus technology.

The purpose of this section is to show how multiple Ant tasks can be scripted to build a complex environment in a relatively short time.

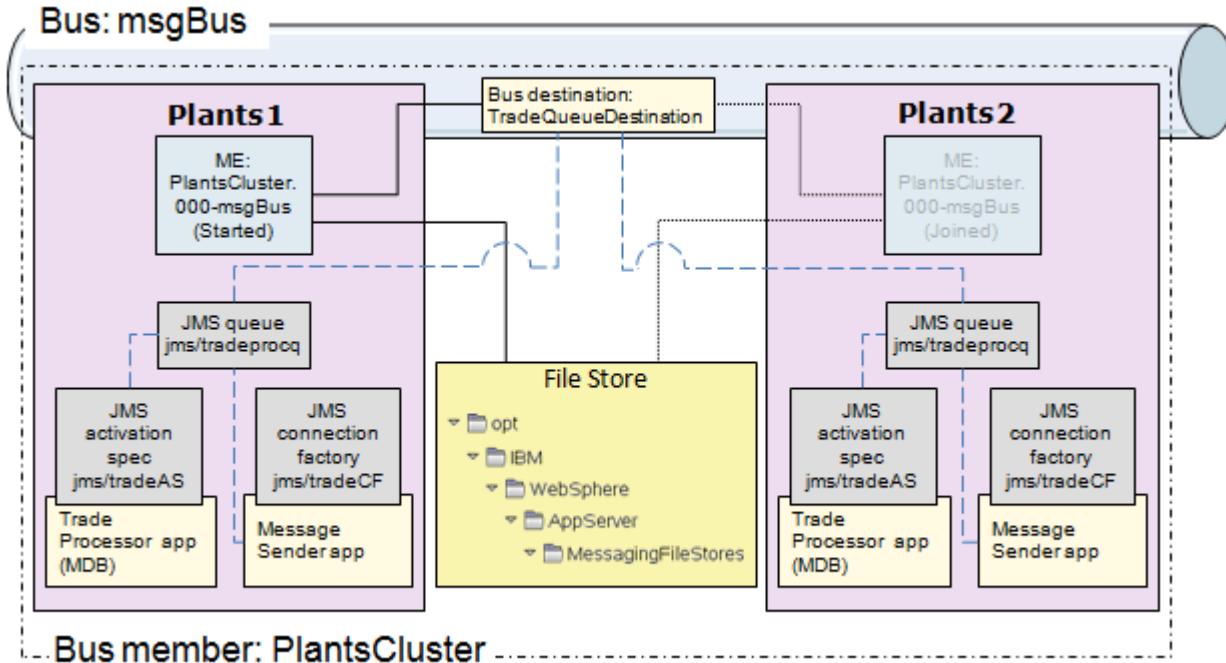
The scripts that you run in this section use the following files:

- A properties file
- A build file
- Several Jython scripts

When you receive applications that use JMS for communication, you must configure the WebSphere environment to support these applications.

The configuration that is required for using the default messaging provider in WebSphere Application Server can be broken down into the following three main parts:

1. Configuring WebSphere Application Server specific SIBus resources
2. Configuring Application-specific JMS resources
3. Installing the applications



In this section, you create the SIBus objects and the JMS objects that are required to support the two messaging applications. You complete the following tasks in this exercise.

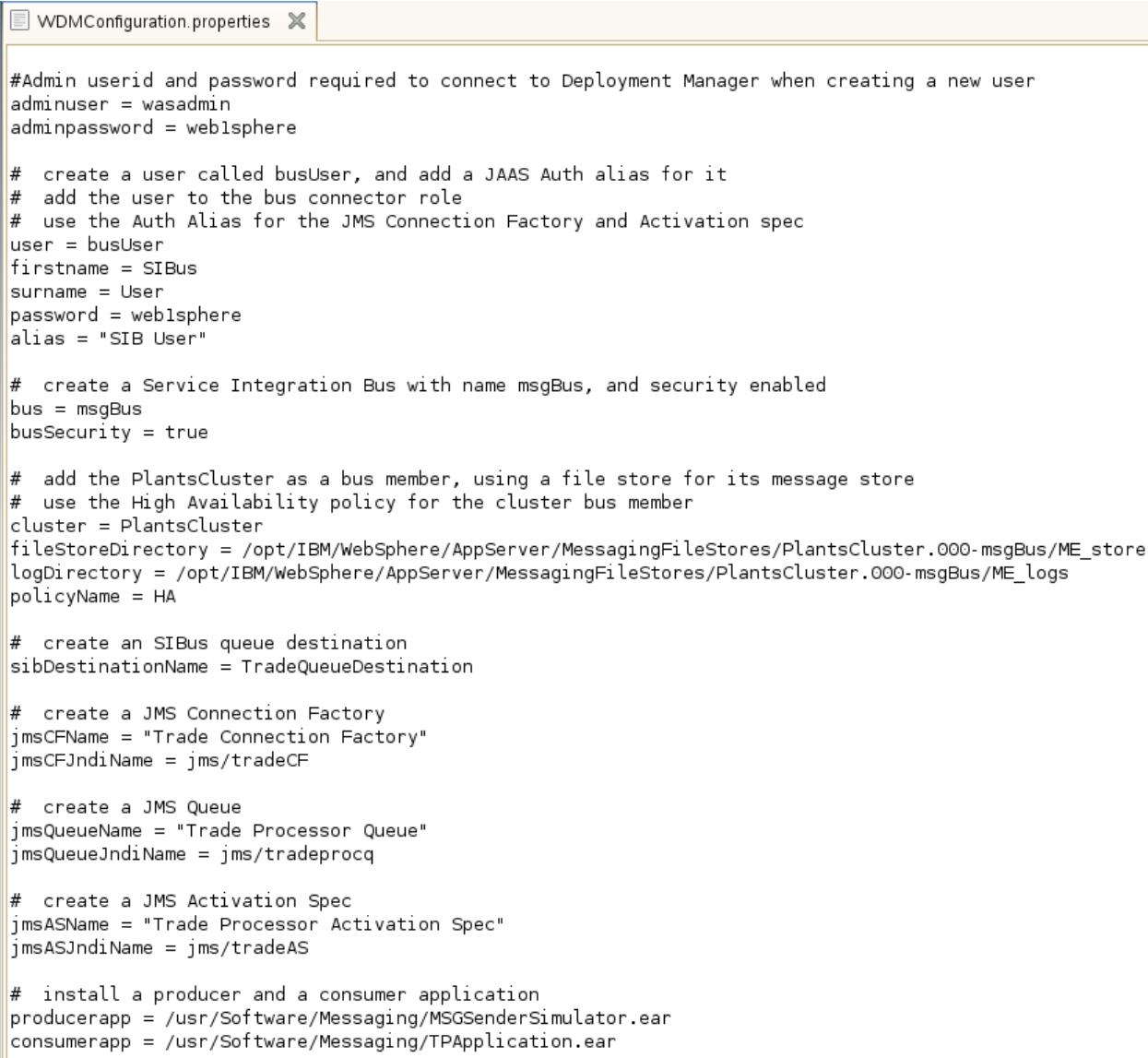
1. Create and secure an SIBus named: `msgBus`
2. Create a bus user and authentication alias, and add the user to the bus connector role.
3. Add the `PlantsCluster` as a member of the bus. As soon as the `PlantsCluster` is a member of the bus, the messaging engine (`PlantsCluster.000-msgBus`) becomes active in one of the cluster members, `Server1`, or `Server2`.
4. Create the bus destination named: `TradeQueueDestination`
5. Create the JMS objects that the messaging applications require: Connection Factory, destination queue, and Activation Specification. These objects are Java objects that are scoped to the cluster and are given JNDI names.
6. Install the messaging applications to the `PlantsCluster`.
7. Test that the applications function properly.

## Examine the scripting artifacts

In this part, you examine the scripting artifacts that are created to complete the default messaging configuration for this exercise.

1. Examine the properties file that contains the values that the scripts use.
  2. Examine the build file that is used to call the scripts.
  3. Examine the scripts that complete the configuration of the environment.
- \_\_\_ 1. Examine the properties file.
- \_\_\_ a. Open a terminal window, and go to `/usr/Software/Messaging/Scripts`

b. Use gedit or vi to open the **WDMConfiguration.properties** file.



The screenshot shows a terminal window with the title "WDMConfiguration.properties". The window contains the following configuration file content:

```
#Admin userid and password required to connect to Deployment Manager when creating a new user
adminuser = wasadmin
adminpassword = weblsphere

# create a user called busUser, and add a JAAS Auth alias for it
# add the user to the bus connector role
# use the Auth Alias for the JMS Connection Factory and Activation spec
user = busUser
firstname = SIBus
surname = User
password = weblsphere
alias = "SIB User"

# create a Service Integration Bus with name msgBus, and security enabled
bus = msgBus
busSecurity = true

# add the PlantsCluster as a bus member, using a file store for its message store
# use the High Availability policy for the cluster bus member
cluster = PlantsCluster
fileStoreDirectory = /opt/IBM/WebSphere/AppServer/MessagingFileStores/PlantsCluster.000-msgBus/ME_store
logDirectory = /opt/IBM/WebSphere/AppServer/MessagingFileStores/PlantsCluster.000-msgBus/ME_logs
policyName = HA

# create an SIBus queue destination
sibDestinationName = TradeQueueDestination

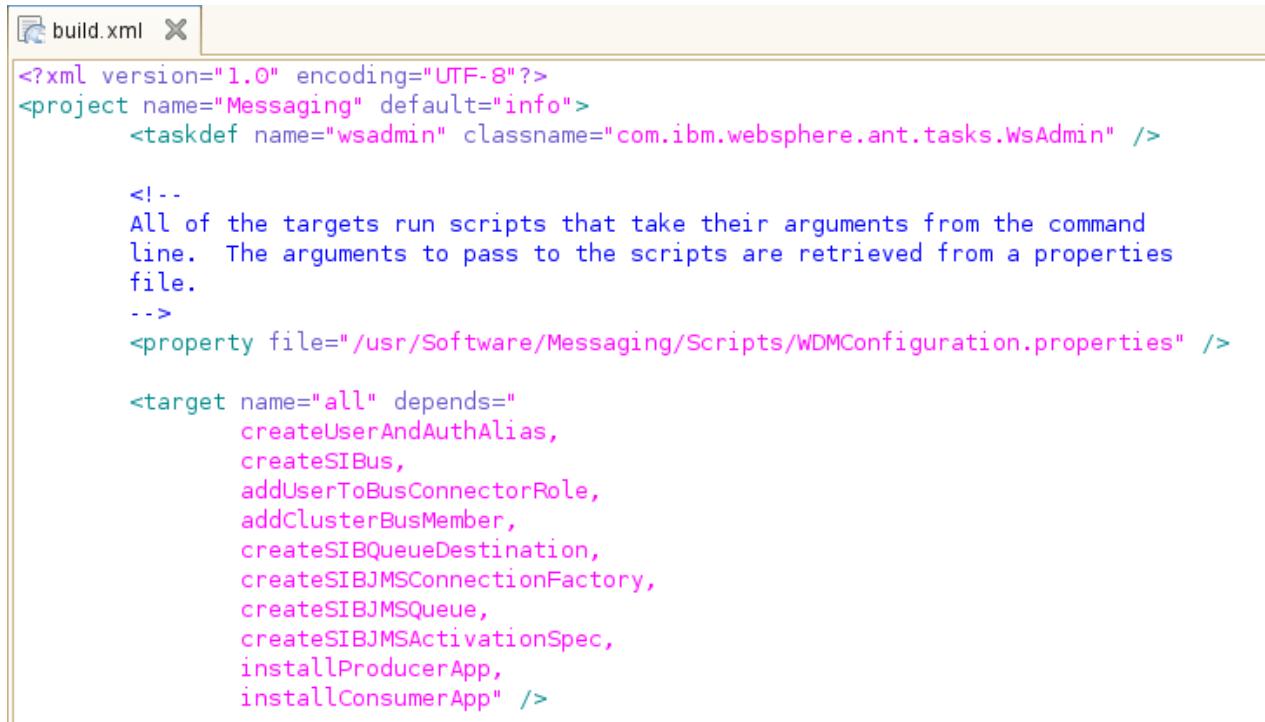
# create a JMS Connection Factory
jmsCFName = "Trade Connection Factory"
jmsCFJndiName = jms/tradeCF

# create a JMS Queue
jmsQueueName = "Trade Processor Queue"
jmsQueueJndiName = jms/tradeprocq

# create a JMS Activation Spec
jmsASName = "Trade Processor Activation Spec"
jmsASJndiName = jms/tradeAS

# install a producer and a consumer application
producerapp = /usr/Software/Messaging/MSGSenderSimulator.ear
consumerapp = /usr/Software/Messaging/TPApplication.ear
```

- \_\_\_ c. Close the **WDMConfiguration.properties** file.
- \_\_\_ 2. Examine the build file.
- \_\_\_ a. Use gedit or vi to open the **build.xml** file.



```

build.xml X

<?xml version="1.0" encoding="UTF-8"?>
<project name="Messaging" default="info">
    <taskdef name="wsadmin" classname="com.ibm.websphere.ant.tasks.WsAdmin" />

    <!--
        All of the targets run scripts that take their arguments from the command
        line. The arguments to pass to the scripts are retrieved from a properties
        file.
    -->
    <property file="/usr/Software/Messaging/Scripts/WDMConfiguration.properties" />

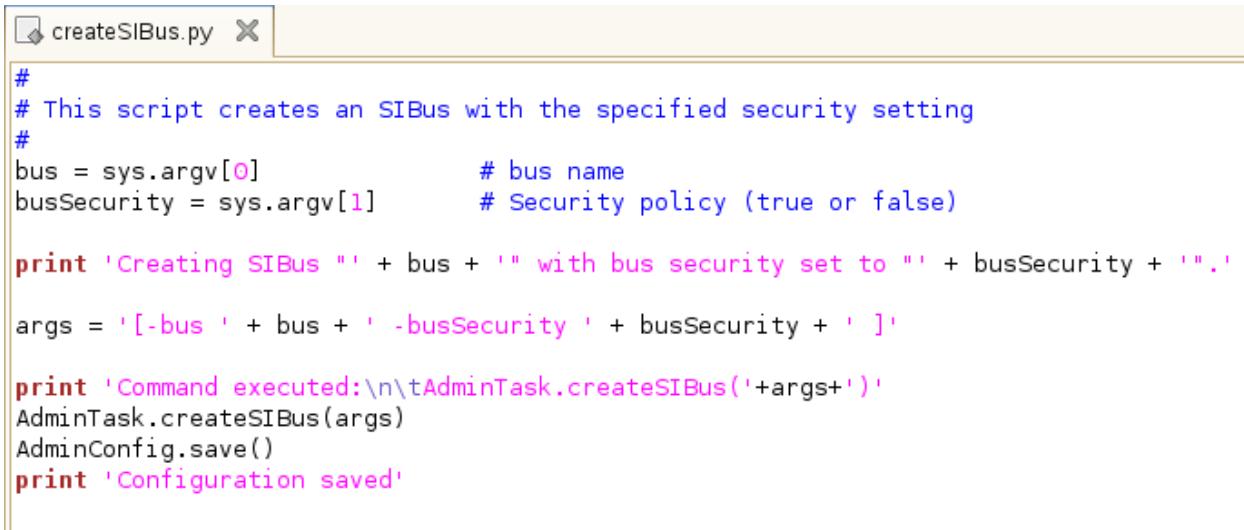
    <target name="all" depends=
            "createUserAndAuthAlias,
             createSIBus,
             addUserToBusConnectorRole,
             addClusterBusMember,
             createSIBQueueDestination,
             createSIBJMSConnectionFactory,
             createSIBJMSQueue,
             createSIBJMSActivationSpec,
             installProducerApp,
             installConsumerApp" />

```

- \_\_\_ b. Note the various tasks that are used to create the configuration. Each task calls a wsadmin script, passing property values as arguments. The properties values used by the tasks are obtained from the **WDMConfiguration.properties** file.
- \_\_\_ c. Scroll down and examine the definitions for all of the listed targets.
- \_\_\_ d. Close the **build.xml** file.

\_\_\_ 3. Examine the Jython scripts.

\_\_\_ a. Use gedit or vi to open the `createSIBus.py` file.



```
#  
# This script creates an SIBus with the specified security setting  
#  
bus = sys.argv[0] # bus name  
busSecurity = sys.argv[1] # Security policy (true or false)  
  
print 'Creating SIBus "' + bus + '" with bus security set to "' + busSecurity + '"'  
args = '[-bus ' + bus + ' -busSecurity ' + busSecurity + ' ]'  
  
print 'Command executed:\n\tAdminTask.createSIBus('+args+')'  
AdminTask.createSIBus(args)  
AdminConfig.save()  
print 'Configuration saved'
```

\_\_\_ b. Notice how relatively simple the script for creating a bus is.

\_\_\_ c. Close the `createSIBus.py` file.

\_\_\_ d. Examine some of the other Jython scripts.

\_\_\_ 4. In the next part of this exercise, you run the scripts to create the service integration bus objects and the JMS resources. The application servers and the node agents must be stopped and only the deployment manager is running.

\_\_\_ a. Log in to the administrative console.

\_\_\_ b. Stop **Plants1** and **Plants2** servers.

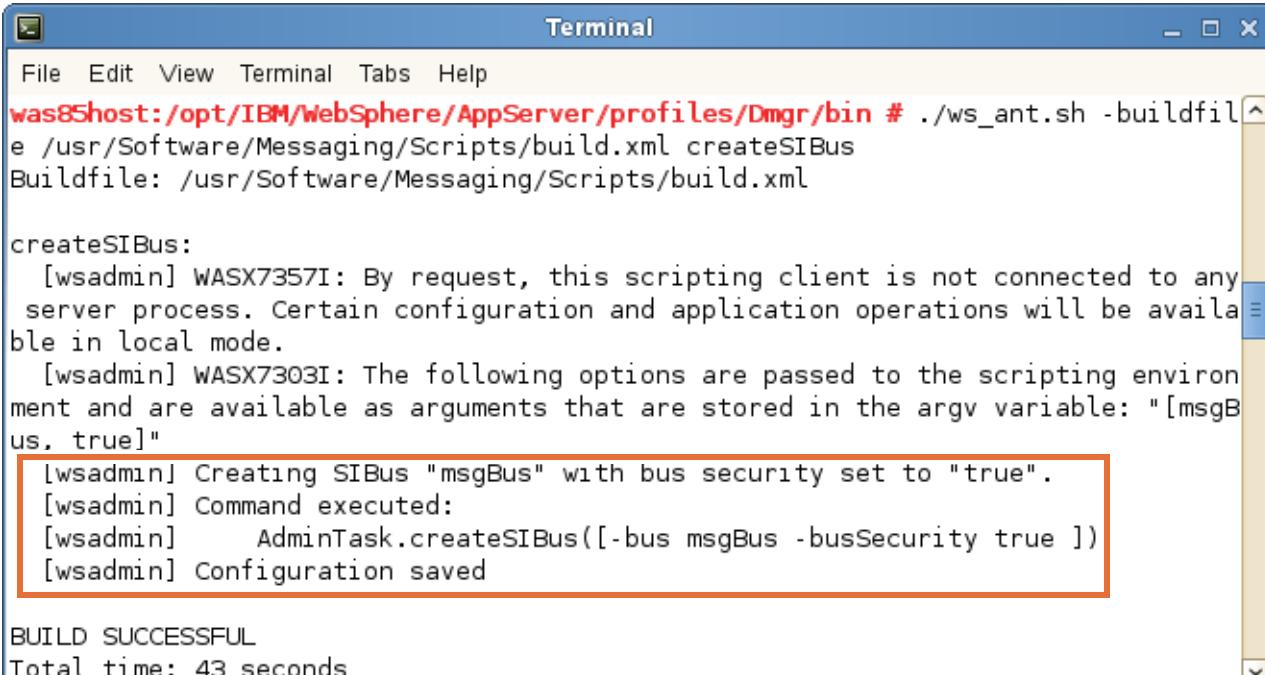
\_\_\_ c. Stop the node agents on **PlantsNode01** and **PlantsNode02**.

\_\_\_ d. Log out of the administrative console.

## Setting up the service integration bus

In this part, you create a secured service integration bus named msgBus. A service integration bus is required to use the WebSphere default messaging provider. The bus is created with security enabled (by setting the busSecurity property to true as required by the design decisions). Since security is enabled, a user with bus connector authority is created, and an authentication alias for that user is created. The applications use the authentication alias to connect to the bus.

- \_\_\_ 1. Create the service integration bus called msgBus with security enabled.
    - \_\_\_ a. Open a terminal window, and go to the `<profile_root>/Dmgr/bin` directory.
    - \_\_\_ b. Enter the following command on one line:
- ```
./ws_ant.sh -buildfile
/usr/Software/Messaging/Scripts/build.xml createSIBus
```



```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -buildfile
e /usr/Software/Messaging/Scripts/build.xml createSIBus
Buildfile: /usr/Software/Messaging/Scripts/build.xml

createSIBus:
[wsadmin] WASX7357I: By request, this scripting client is not connected to any
server process. Certain configuration and application operations will be availa
ble in local mode.
[wsadmin] WASX7303I: The following options are passed to the scripting environ
ment and are available as arguments that are stored in the argv variable: "[msgB
us, true]"
[wsadmin] Creating SIBus "msgBus" with bus security set to "true".
[wsadmin] Command executed:
[wsadmin]     AdminTask.createSIBus([-bus msgBus -busSecurity true ])
[wsadmin] Configuration saved

BUILD SUCCESSFUL
Total time: 43 seconds
```

- \_\_\_ c. Make sure that the build completes successfully, and that bus **msgBus** is created with security enabled.
- \_\_\_ 2. Create the bus user and authentication alias.
  - \_\_\_ a. Enter the following command on one line:

```
./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml  
createUserAndAuthAlias
```

The terminal window shows the execution of the `ws_ant.sh` script to create a user and authentication alias. The output indicates a successful connection to the Deployment Manager and the creation of a user named `busUser` and an authentication alias named `SIB User`. The entire command execution is highlighted with a red box.

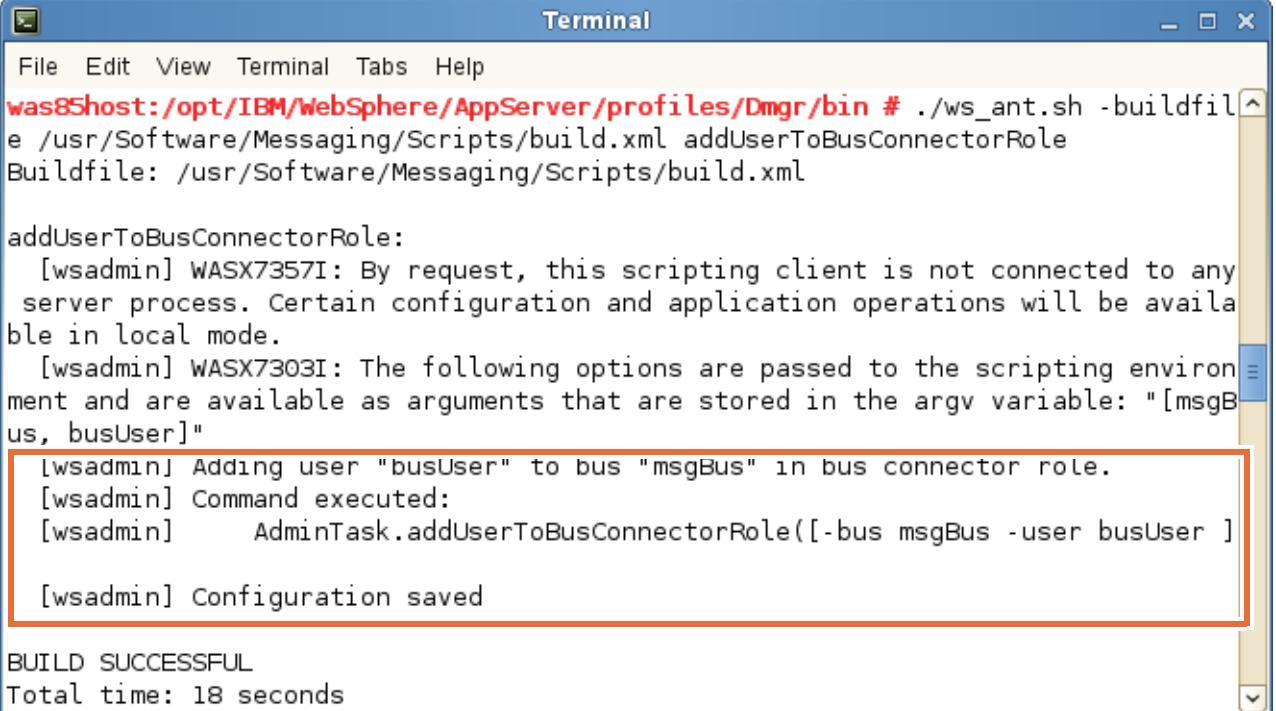
```
File Edit View Terminal Tabs Help  
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml createUserAndAuthAlias  
Buildfile: /usr/Software/Messaging/Scripts/build.xml  
  
createUserAndAuthAlias:  
[wsadmin] WASX7209I: Connected to process "dmgr" on node dmgrNode using SOAP connector; The type of process is: DeploymentManager  
[wsadmin] WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable: "[busUser. websphere. SIBus. User. \"SIB User\"]"  
[wsadmin] Creating user "busUser" (SIBus User).  
[wsadmin] Command executed:  
[wsadmin] AdminTask.createUser([-uid busUser -password websphere -cn SIBUs -sn User -confirmPassword websphere ])  
[wsadmin] Creating Auth Alias ""SIB User"" for user "busUser".  
[wsadmin] Command executed:  
[wsadmin] AdminTask.createAuthDataEntry([-alias "SIB User" -user busUser -password websphere ])  
[wsadmin] Configuration saved  
  
BUILD SUCCESSFUL  
Total time: 35 seconds
```

- \_\_ b. Make sure that the build completes successfully and the user **busUser** and authentication alias **SIB User** are created.

\_\_\_ 3. Add the user to the bus connector security role.

\_\_\_ a. Enter the following command on one line:

```
./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml
addUserToBusConnectorRole
```



The screenshot shows a terminal window titled "Terminal". The command entered was `./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml addUserToBusConnectorRole`. The output shows the buildfile path, the command being run, and the wsadmin logs. A red box highlights the successful addition of the user to the bus connector role, followed by a message about configuration being saved and the build being successful.

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -buildfil
e /usr/Software/Messaging/Scripts/build.xml addUserToBusConnectorRole
Buildfile: /usr/Software/Messaging/Scripts/build.xml

addUserToBusConnectorRole:
[wsadmin] WASX7357I: By request, this scripting client is not connected to any
server process. Certain configuration and application operations will be availa
ble in local mode.
[wsadmin] WASX7303I: The following options are passed to the scripting environm
ent and are available as arguments that are stored in the argv variable: "[msgB
us, busUser]"
[wsadmin] Adding user "busUser" to bus "msgBus" in bus connector role.
[wsadmin] Command executed:
[wsadmin]     AdminTask.addUserToBusConnectorRole([-bus msgBus -user busUser ])

[wsadmin] Configuration saved

BUILD SUCCESSFUL
Total time: 18 seconds
```

\_\_\_ b. Make sure that the build completes successfully and that **busUser** is added to the **bus connector role**.

\_\_\_ 4. Add the cluster as a bus member.

\_\_\_ a. Enter the following command on one line:

```
./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml  
addClusterBusMember
```

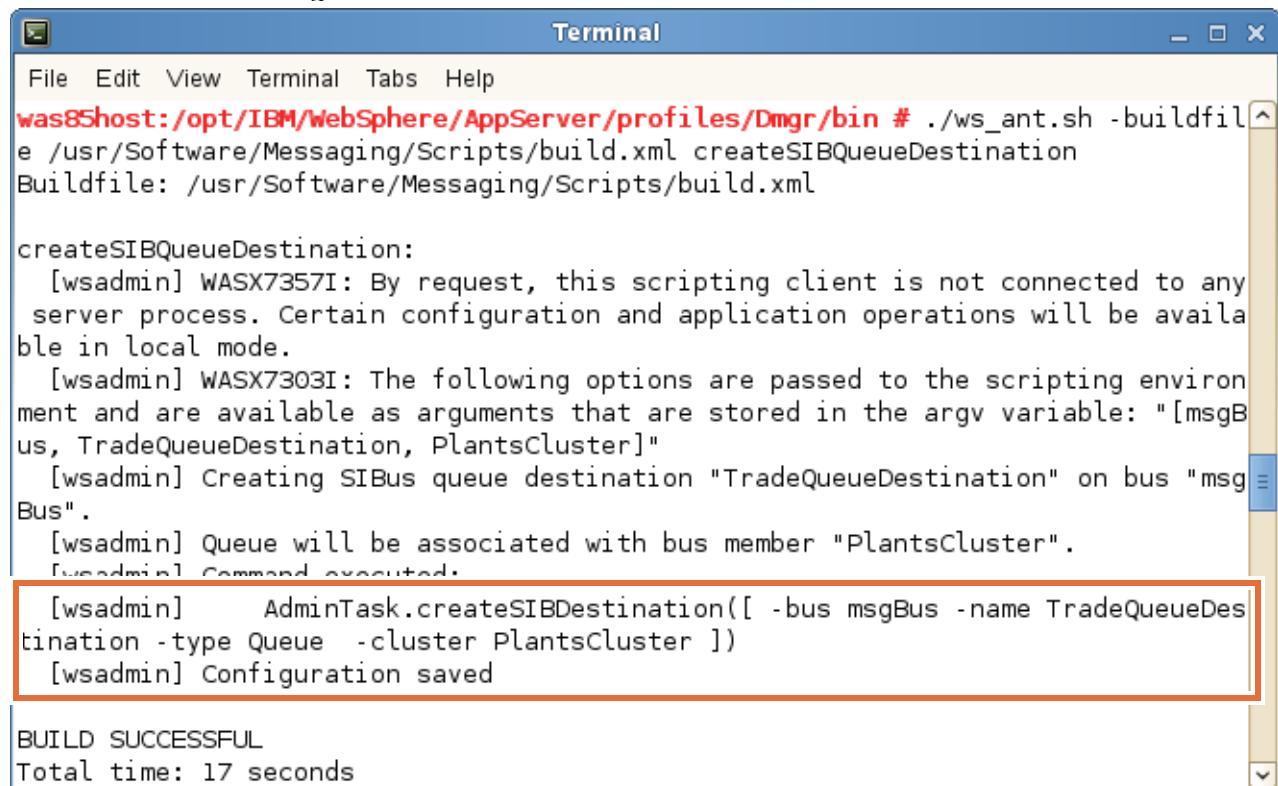
The screenshot shows a terminal window with the title "Terminal". The window contains the following text:

```
File Edit View Terminal Tabs Help  
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml addClusterBusMember  
Buildfile: /usr/Software/Messaging/Scripts/build.xml  
  
addClusterBusMember:  
[wsadmin] WASX7357I: By request, this scripting client is not connected to any server process. Certain configuration and application operations will be available in local mode.  
[wsadmin] WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable: "[msgBus, PlantsCluster, /opt/IBM/WebSphere/AppServer/MessagingFileStores/PlantsCluster.000-msgBus/ME_store, /opt/IBM/WebSphere/AppServer/MessagingFileStores/PlantsCluster.000-msgBus/ME_logs, HA]"  
[wsadmin] Adding cluster "PlantsCluster" as bus member to bus "msgBus" with "HA" policy.  
[wsadmin] Using "/opt/IBM/WebSphere/AppServer/MessagingFileStores/PlantsCluster.000-msgBus/ME_store" for permanent and temporary store.  
[wsadmin] Using "/opt/IBM/WebSphere/AppServer/MessagingFileStores/PlantsCluster.000-msgBus/ME_logs" for logs.  
[wsadmin] Deletes old message stores if this bus member had been added previously.  
[wsadmin] Command executed:  
[wsadmin] AdminTask.addSIBusMember({ -bus msgBus -cluster PlantsCluster -fileStore -permanentStoreDirectory /opt/IBM/WebSphere/AppServer/MessagingFileStores/PlantsCluster.000-msgBus/ME_store -logDirectory /opt/IBM/WebSphere/AppServer/MessagingFileStores/PlantsCluster.000-msgBus/ME_logs -temporaryStoreDirectory /opt/IBM/WebSphere/AppServer/MessagingFileStores/PlantsCluster.000-msgBus/ME_store -enableAssistance true -policyName HA })  
[wsadmin] Configuration saved  
  
BUILD SUCCESSFUL  
Total time: 37 seconds
```

\_\_\_ b. Make sure that the build completes successfully and the **PlantsCluster** is added as a member of **msgBus**.

- \_\_\_ 5. Create the queue destination on the bus.
- \_\_\_ a. Enter the following command on one line:

```
./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml
createSIBQueueDestination
```



The screenshot shows a terminal window titled "Terminal". The command entered was `./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml createSIBQueueDestination`. The output shows the buildfile path, the creation of the queue destination, and a warning about local mode. It then lists options for the scripting environment, shows the creation of the queue destination on the msgBus, and associates it with the PlantsCluster. The final output indicates a successful build and a total time of 17 seconds.

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -buildfil
e /usr/Software/Messaging/Scripts/build.xml createSIBQueueDestination
Buildfile: /usr/Software/Messaging/Scripts/build.xml

createSIBQueueDestination:
[wsadmin] WASX7357I: By request, this scripting client is not connected to any
server process. Certain configuration and application operations will be availa
ble in local mode.
[wsadmin] WASX7303I: The following options are passed to the scripting environ
ment and are available as arguments that are stored in the argv variable: "[msgB
us, TradeQueueDestination, PlantsCluster]"
[wsadmin] Creating SIBus queue destination "TradeQueueDestination" on bus "msg
Bus".
[wsadmin] Queue will be associated with bus member "PlantsCluster".
[wsadmin] Command executed.
[wsadmin] AdminTask.createSIBDestination([ -bus msgBus -name TradeQueueDes
tination -type Queue -cluster PlantsCluster ])
[wsadmin] Configuration saved

BUILD SUCCESSFUL
Total time: 17 seconds
```

- \_\_\_ b. Make sure that the build completes successfully and the queue **TradeQueueDestination** is added to the bus.
- \_\_\_ 6. All of the necessary service integration bus objects are created. Use the administrative console to examine the objects that you created.
- \_\_\_ a. Log in to the administrative console.

- \_\_\_ b. Click **Service integration > buses** to see the bus. Notice that security is enabled.

| Select                   | Name                   | Description | Security                |
|--------------------------|------------------------|-------------|-------------------------|
| <input type="checkbox"/> | <a href="#">msgBus</a> |             | <a href="#">Enabled</a> |

Total 1

- \_\_\_ c. By clicking the bus name, msgBus, you can find most of the objects you created. Look for the messaging engine (ME) under **msgBus > Bus members > PlantsCluster**.
- \_\_\_ d. Look for the queue destination under **msgBus > Destinations**.
- \_\_\_ e. The user and authentication alias are not under Service integration. Look for the busUser under **Users and Groups > Manage Users**.
- \_\_\_ f. Look for the authentication alias, SIB\_User, under **Security > Global security > Java Authentication and Authorization Service > J2C authentication data**.
- \_\_\_ g. Log out of the administrative console.

## Configuring the JMS resources

Now that the service integration bus is configured; it is necessary to configure the JMS resources so that the applications can produce and use messages.

There are three JMS resources that you must configure for the applications to work; they are:

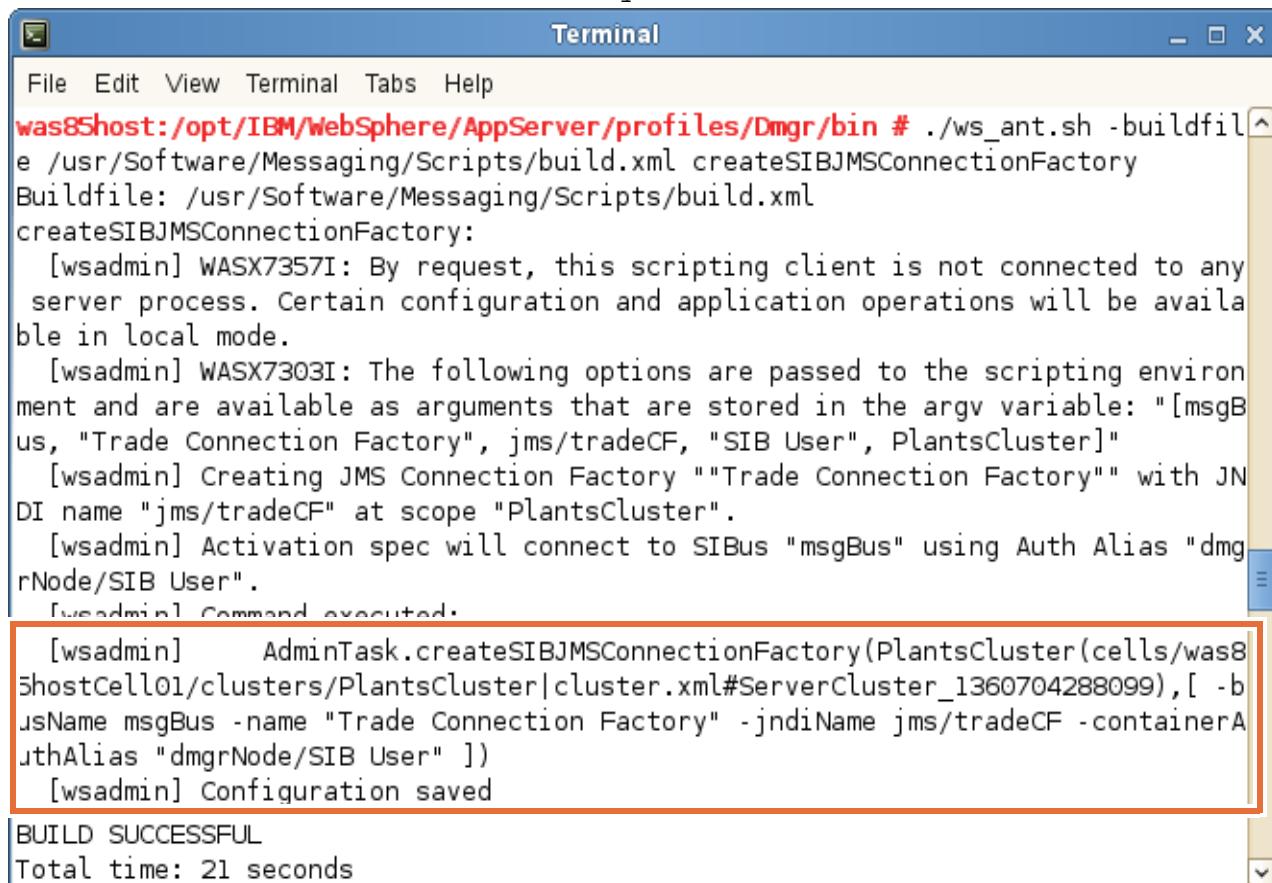
- JMS Connection Factory
- JMS Queue
- JMS Activation Specification

Remember that applications have no knowledge of the service integration bus itself. The applications use JMS to place and retrieve messages that use, in this case, a JMS Queue.

\_\_\_ 1. Create the JMS Connection Factory.

\_\_\_ a. Enter the following command on one line:

```
./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml
createSIBJMSConnectionFactory
```



The screenshot shows a terminal window titled "Terminal". The command entered is `./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml createSIBJMSConnectionFactory`. The output shows the buildfile path, the creation of the connection factory, and several informational messages from the wsadmin scripting client. A red box highlights the configuration command and its output, which includes the JNDI name and authentication alias. The terminal also shows a successful build and the total execution time.

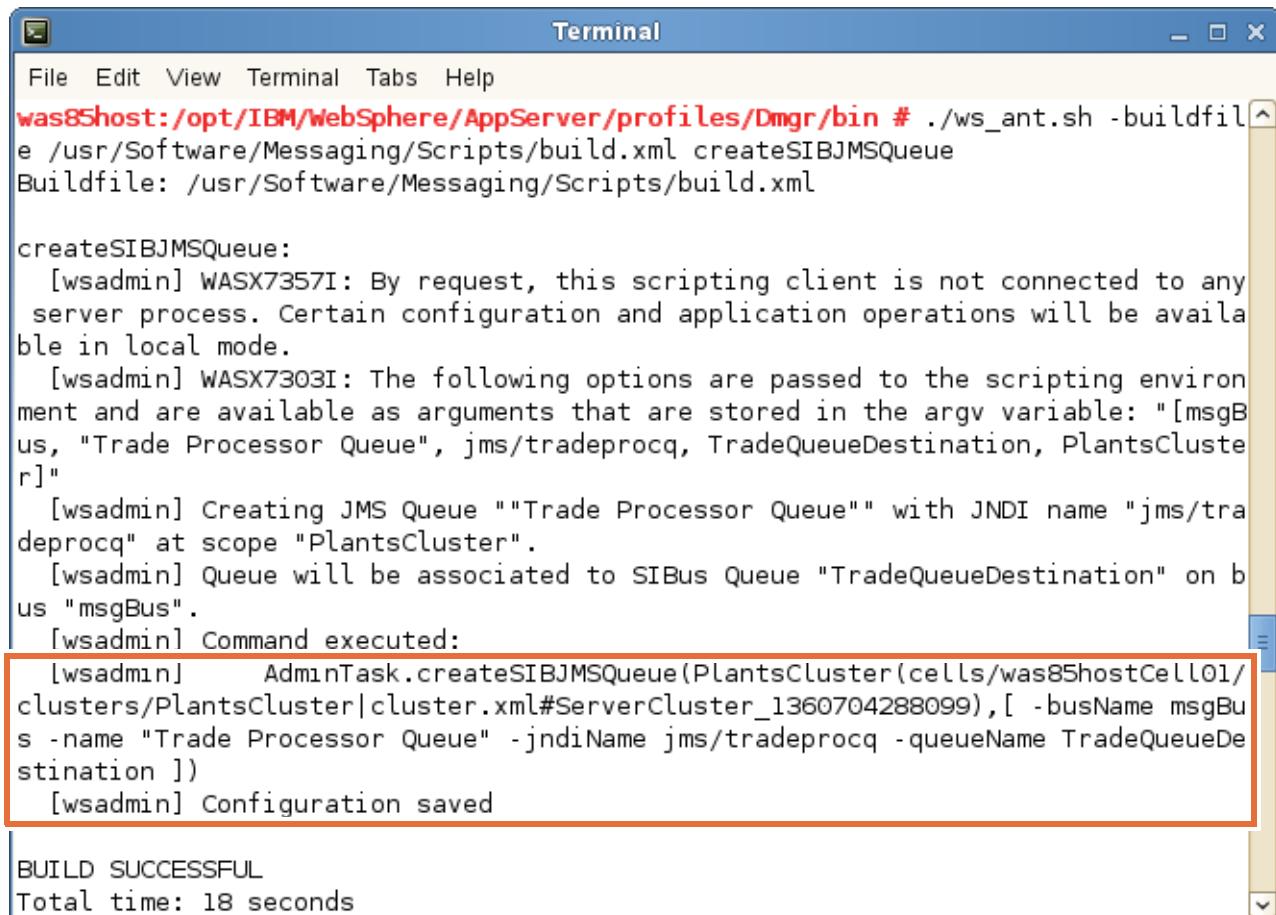
```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml createSIBJMSConnectionFactory
Buildfile: /usr/Software/Messaging/Scripts/build.xml
createSIBJMSConnectionFactory:
[wsadmin] WASX7357I: By request, this scripting client is not connected to any server process. Certain configuration and application operations will be available in local mode.
[wsadmin] WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable: "[msgBus, "Trade Connection Factory", jms/tradeCF, "SIB User", PlantsCluster]"
[wsadmin] Creating JMS Connection Factory "Trade Connection Factory" with JNDI name "jms/tradeCF" at scope "PlantsCluster".
[wsadmin] Activation spec will connect to SIBus "msgBus" using Auth Alias "dmgrNode/SIB User".
[wsadmin] Command executed.
[wsadmin] AdminTask.createSIBJMSConnectionFactory(PlantsCluster(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#ServerCluster_1360704288099),[-b
jsName msgBus -name "Trade Connection Factory" -jndiName jms/tradeCF -containerA
uthAlias "dmgrNode/SIB User"])
[wsadmin] Configuration saved
BUILD SUCCESSFUL
Total time: 21 seconds
```

\_\_\_ b. Make sure that the build completes successfully and the **Trade Connection Factory** is created.

\_\_ 2. Create the JMS Queue.

\_\_ a. Enter the following command on one line:

```
./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml  
createSIBJMSQueue
```



The terminal window shows the execution of the command `./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml createSIBJMSQueue`. The output indicates that the buildfile is located at `/usr/Software/Messaging/Scripts/build.xml`. The process starts by defining the `createSIBJMSQueue` target. It then handles local mode configuration and provides options for the scripting environment. The command successfully creates a JMS Queue named "Trade Processor Queue" with JNDI name "jms/tradeprocq" under the PlantsCluster bus. The configuration command is shown as `AdminTask.createSIBJMSQueue(PlantsCluster(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#ServerCluster_1360704288099), [-busName msgBus -name "Trade Processor Queue" -jndiName jms/tradeprocq -queueName TradeQueueDestination])`. The configuration is saved, and the build is successful, taking 18 seconds.

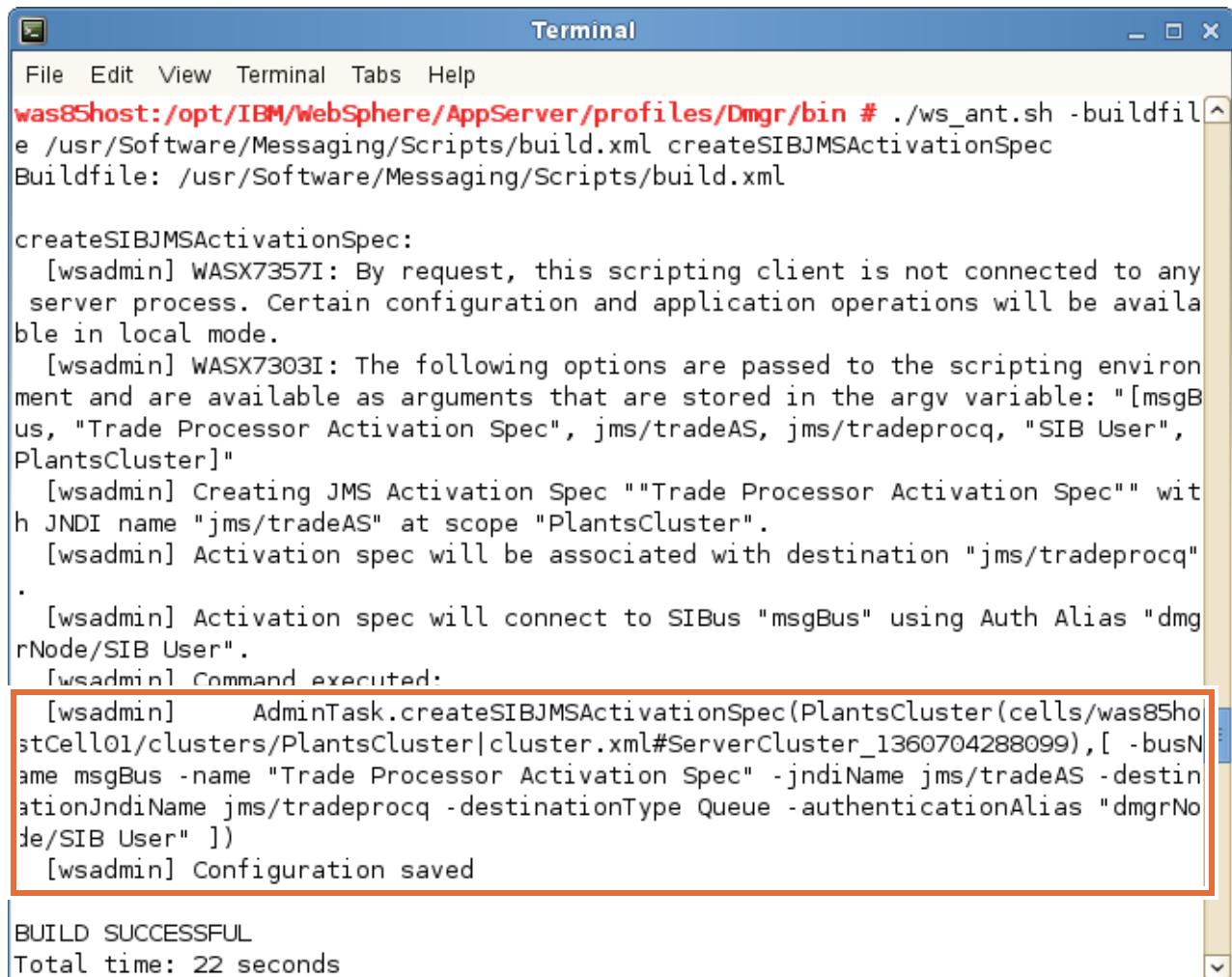
```
Terminal  
File Edit View Terminal Tabs Help  
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml createSIBJMSQueue  
Buildfile: /usr/Software/Messaging/Scripts/build.xml  
  
createSIBJMSQueue:  
[wsadmin] WASX7357I: By request, this scripting client is not connected to any server process. Certain configuration and application operations will be available in local mode.  
[wsadmin] WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable: "[msgBus, "Trade Processor Queue", jms/tradeprocq, TradeQueueDestination, PlantsCluster]"  
[wsadmin] Creating JMS Queue ""Trade Processor Queue"" with JNDI name "jms/tradeprocq" at scope "PlantsCluster".  
[wsadmin] Queue will be associated to SIBus Queue "TradeQueueDestination" on bus "msgBus".  
[wsadmin] Command executed:  
[wsadmin] AdminTask.createSIBJMSQueue(PlantsCluster(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#ServerCluster_1360704288099), [-busName msgBus -name "Trade Processor Queue" -jndiName jms/tradeprocq -queueName TradeQueueDestination])  
[wsadmin] Configuration saved  
  
BUILD SUCCESSFUL  
Total time: 18 seconds
```

\_\_ b. Make sure that the build completes successfully and the **Trade Processor Queue** is created.

\_\_\_ 3. Create the JMS Activation Specification.

\_\_\_ a. Enter the following command on one line:

```
./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml
createsSIBJMSActivationSpec
```



The screenshot shows a terminal window titled "Terminal". The command entered was `./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml createsSIBJMSActivationSpec`. The output shows the build file being identified as `/usr/Software/Messaging/Scripts/build.xml`, and the activation specification being created with the name `Trade Processor Activation Spec`. The configuration details include connecting to SIBus "msgBus" using Auth Alias "dmgrNode/SIB User". A red box highlights the command `AdminTask.createSIBJMSActivationSpec(PlantsCluster(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#ServerCluster_1360704288099),[ -busName msgBus -name "Trade Processor Activation Spec" -jndiName jms/tradeAS -destinationJndiName jms/tradeprocq -destinationType Queue -authenticationAlias "dmgrNode/SIB User" ])` and its confirmation message "Configuration saved". The terminal also displays "BUILD SUCCESSFUL" and a total time of 22 seconds.

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin # ./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml createsSIBJMSActivationSpec
Buildfile: /usr/Software/Messaging/Scripts/build.xml

createSIBJMSActivationSpec:
[wsadmin] WASX7357I: By request, this scripting client is not connected to any server process. Certain configuration and application operations will be available in local mode.
[wsadmin] WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable: "[msgBus, "Trade Processor Activation Spec", jms/tradeAS, jms/tradeprocq, "SIB User", PlantsCluster]"
[wsadmin] Creating JMS Activation Spec "Trade Processor Activation Spec" with JNDI name "jms/tradeAS" at scope "PlantsCluster".
[wsadmin] Activation spec will be associated with destination "jms/tradeprocq"
.
[wsadmin] Activation spec will connect to SIBus "msgBus" using Auth Alias "dmgrNode/SIB User".
[wsadmin] Command executed.
[wsadmin] AdminTask.createSIBJMSActivationSpec(PlantsCluster(cells/was85hostCell01/clusters/PlantsCluster|cluster.xml#ServerCluster_1360704288099),[ -busName msgBus -name "Trade Processor Activation Spec" -jndiName jms/tradeAS -destinationJndiName jms/tradeprocq -destinationType Queue -authenticationAlias "dmgrNode/SIB User" ])
[wsadmin] Configuration saved

BUILD SUCCESSFUL
Total time: 22 seconds
```

\_\_\_ b. Make sure that the build completes successfully and the **Trade Processor Activation Spec** is created.

\_\_\_ 4. Use the administrative console to examine the JMS objects that you created.

\_\_\_ a. Log in to the administrative console.

- \_\_\_ b. All of the JMS resources that you created are under **Resources > JMS**.



- \_\_\_ c. Click **Connection factories** to see the Trade Connection Factory.  
 \_\_\_ d. Click **Queues** to see the Trade Processor Queue.  
 \_\_\_ e. Click **Activation specifications** to see the Trade Processor Activation Spec.  
 \_\_\_ f. When you are done examining the JMS resources, log out of the administrative console.

## Install the messaging applications

As mentioned at the beginning of this section, two applications are used to demonstrate messaging on the service integration bus. The first application produces messages that are placed on a queue, and the second application uses those messages. Now that the message bus and the JMS resources are configured, you must install the applications.

- \_\_\_ 1. Install the MSGSenderSimulator application. Since this application is writing messages to the bus, it must know how to find the Connection Factory that was previously defined. It also must know how to find the JMS Queue. The application uses references to find the JNDI name for the Connection Factory and queue. These references must be assigned when the application is installed.

- \_\_\_ a. Enter the following command on one line:

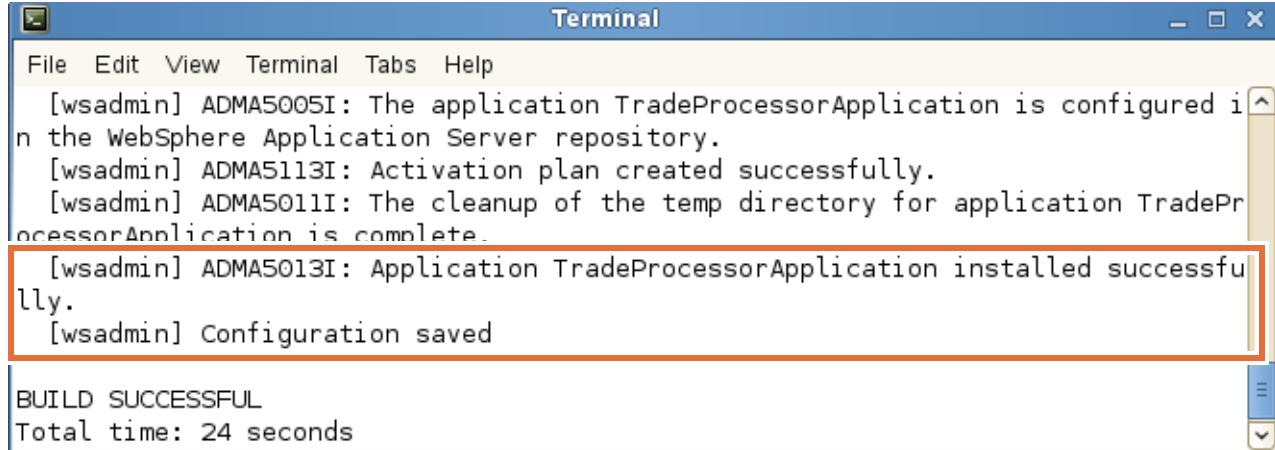
```
./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml  
installProducerApp
```

The terminal window shows the execution of the command. The output indicates that the application was configured and activated successfully, and then installed successfully. The line '[wsadmin] ADMA5013I: Application MSGSenderSimulator installed successfully.' is highlighted with a red rectangle.

```
File Edit View Terminal Tabs Help  
[wsadmin] ADMA5005I: The application MSGSenderSimulator is configured in the WebSphere Application Server repository.  
[wsadmin] ADMA5113I: Activation plan created successfully.  
[wsadmin] ADMA5011I: The cleanup of the temp directory for application MSGSenderSimulator is complete  
[wsadmin] ADMA5013I: Application MSGSenderSimulator installed successfully.  
[wsadmin] Configuration saved  
  
BUILD SUCCESSFUL  
Total time: 30 seconds
```

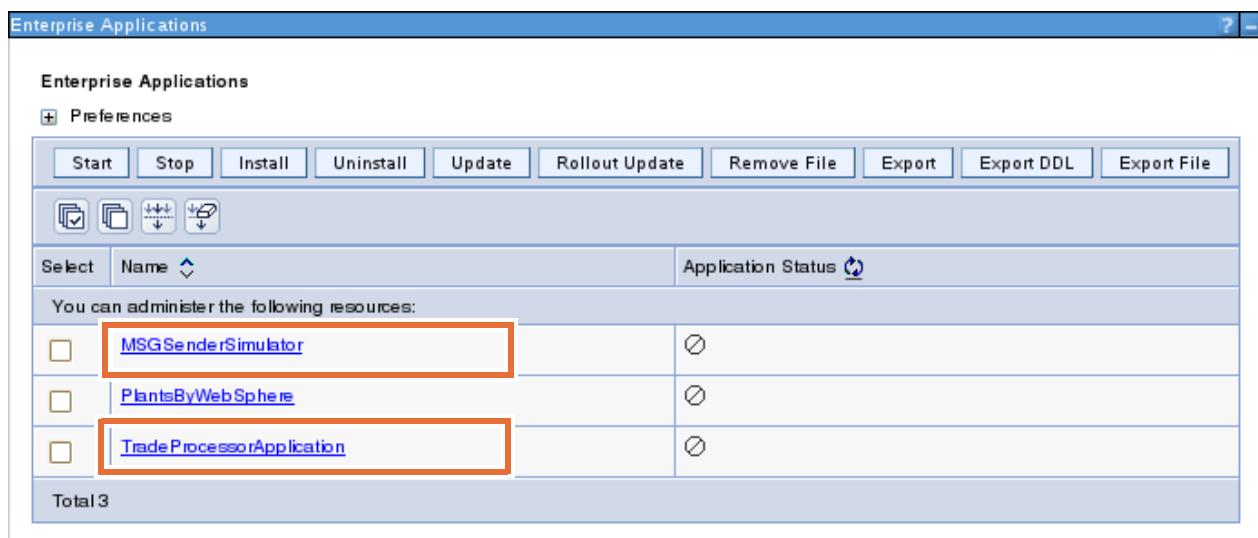
- \_\_\_ b. Make sure that the build completes successfully and the **MSGSenderSimulator** application is installed.
- \_\_\_ 2. Install the Trade processor application. This application uses the messages, so it must know how to find the Activation Specification (already coded into the application).
- \_\_\_ a. Enter the following command on one line:

```
./ws_ant.sh -buildfile /usr/Software/Messaging/Scripts/build.xml
installConsumerApp
```



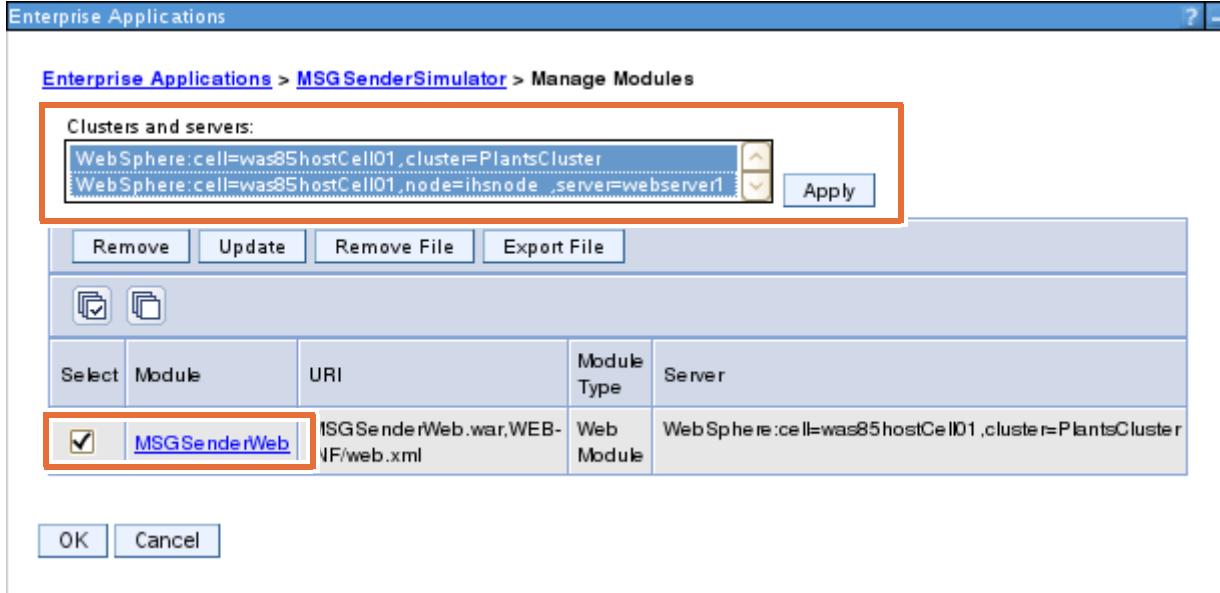
```
Terminal
File Edit View Terminal Tabs Help
[wsadmin] ADMA5005I: The application TradeProcessorApplication is configured in the WebSphere Application Server repository.
[wsadmin] ADMA5113I: Activation plan created successfully.
[wsadmin] ADMA5011I: The cleanup of the temp directory for application TradeProcessorApplication is complete.
[wsadmin] ADMA5013I: Application TradeProcessorApplication installed successfully.
[wsadmin] Configuration saved
BUILD SUCCESSFUL
Total time: 24 seconds
```

- \_\_\_ b. Make sure that the build completes successfully and the **TradeProcessorApplication** is installed.
- \_\_\_ 3. Use the administrative console to verify that the applications are installed.
- \_\_\_ a. Log in to the administrative console.
- \_\_\_ b. Click **Applications > Application Types > WebSphere enterprise applications**.



| Enterprise Applications  |                            |
|--------------------------|----------------------------|
| <input type="checkbox"/> | MSGSenderSimulator         |
| <input type="checkbox"/> | PlantsByWebSphere          |
| <input type="checkbox"/> | Trade ProcessorApplication |
| Total 3                  |                            |

- \_\_\_ 4. Map the web components of the messaging applications to the web server and the PlantsCluster. This step can be scripted as you did with the PlantsByWebSphere application. You can also use the administrative console, and command assistance.
- \_\_\_ a. Click the **MSGSenderSimulator** link, and then click **Manages Modules**.
- \_\_\_ b. Check the module, select both **PlantsCluster** and **webserver1**, and click **Apply**.



- \_\_\_ c. Verify that both **PlantsCluster** and **webserver1** are listed in the Server column.

| Remove                   | Update                       | Remove File                      | Export File |                                                                                                                        |
|--------------------------|------------------------------|----------------------------------|-------------|------------------------------------------------------------------------------------------------------------------------|
| Select                   | Module                       | URI                              | Module Type | Server                                                                                                                 |
| <input type="checkbox"/> | <a href="#">MSGSenderWeb</a> | MSGSenderWeb.war,WEB-INF/web.xml | Web Module  | WebSphere:cell=was85hostCell01,node=ihsnode ,server=webserver1<br>WebSphere:cell=was85hostCell01,cluster=PlantsCluster |
| <input type="checkbox"/> | <a href="#">MSGSenderWeb</a> | MSGSenderWeb.war,WEB-INF/web.xml | Web Module  | WebSphere:cell=was85hostCell01,node=ihsnode ,server=webserver1<br>WebSphere:cell=was85hostCell01,cluster=PlantsCluster |

- \_\_\_ d. Click **OK**. You might want to click the **View administrative scripting command for the last action** link in the Help portlet to see the Jython commands.
- \_\_\_ e. Save the changes.
- \_\_\_ f. Back on the Enterprise applications pane, click **TradeProcessorApplication > Manage Modules**.

- \_\_\_ g. Check only the **TradeProcessorWeb** module, select both **PlantsCluster** and **webserver1**, and click **Apply**.

| Select                              | Module             | URI                                    | Module Type | Server                                               |
|-------------------------------------|--------------------|----------------------------------------|-------------|------------------------------------------------------|
| <input type="checkbox"/>            | TPEJB              | TPEJB.jar,META-INF/ejb-jar.xml         | EJB Module  | WebSphere:cell=was85hostCell01,cluster=PlantsCluster |
| <input checked="" type="checkbox"/> | Trade ProcessorWeb | Trade ProcessorWeb.war,WEB-INF/web.xml | Web Module  | WebSphere:cell=was85hostCell01,cluster=PlantsCluster |

- \_\_\_ h. Verify that both **PlantsCluster** and **webserver1** are listed in the Server column.
- \_\_\_ i. Click **OK**. Again, you might want to click the **View administrative scripting command for the last action** link in the Help portlet to see the Jython commands.
- \_\_\_ j. Save the changes.



### Information

## Administrative scripting commands

Here are the scripting commands that the administrative console generates for the module mapping that you did.

```
AdminApp.edit('MSGSenderSimulator', '[ -MapModulesToServers [[
MSGSenderWeb MSGSenderWeb.war,WEB-INF/web.xml
WebSphere:cell=was85hostCell01,cluster=PlantsCluster+WebSphere:cell=was85hostCell01,node=ihsnode,server=webserver1 ]]]' )
```

```
AdminApp.edit('TradeProcessorApplication', '[ -MapModulesToServers [[
TradeProcessorWeb TradeProcessorWeb.war,WEB-INF/web.xml
WebSphere:cell=was85hostCell01,cluster=PlantsCluster+WebSphere:cell=was85hostCell01,node=ihsnode,server=webserver1 ]]]' )
```

- \_\_\_ k. Log out of the administrative console.

## Start the web server

- \_\_\_ 1. Open a terminal window, and go to `/opt/IBM/HTTPServer/bin`
- \_\_\_ 2. Enter the following commands:

```
./apachectl start  
./adminctl start
```

## Start the applications and verify that they run properly

- \_\_\_ 1. Start the node agents and the PlantsCluster.
  - \_\_\_ a. Open a terminal window, and go to `<profile_root>/PlantsProfile1/bin`
  - \_\_\_ b. Enter the command: `./startNode.sh`
  - \_\_\_ c. Open a terminal window, and go to `<profile_root>/PlantsProfile2/bin`
  - \_\_\_ d. Enter the command: `./startNode.sh`
  - \_\_\_ e. Log in to the administrative console
  - \_\_\_ f. Click **Servers > Clusters > WebSphere application server clusters**.
  - \_\_\_ g. Select **PlantsCluster**, and click **Start**.
  - \_\_\_ h. Wait for the cluster to start.

2. Start the Monitor HTML page. The web page is provided as part of the Trade processor application to more easily demonstrate the interactions between the application servers and messaging engines.
- a. Open a new browser and enter the web address:  
<http://was85host/Trade/processor/Monitor.html>

The screenshot shows a browser window titled "Monitor" with two main panes. The left pane is titled "Sending messages from server: Plants1" and contains instructions to select the number of messages for Buy and Sell categories and click "Send messages". It has input fields for "Buy messages" (set to 1) and "Sell messages" (set to 1), both with up/down arrows, and a "Send messages" button. The right pane is titled "Trade Requests To Be Processed On Server: Plants1" and features a header "Refresh every: 10 seconds" and a table with columns: Account, Buy/Sell, Symbol, Qty, Total cost, Transaction. The right pane is currently empty. A vertical scroll bar is visible between the two panes.

The left pane is titled "Sending messages from server: Plants2" and contains similar instructions. It also has input fields for "Buy messages" (set to 1) and "Sell messages" (set to 1), both with up/down arrows, and a "Send messages" button. The right pane is titled "Trade Requests To Be Processed On Server: Plants2" and features a header "Refresh every: 10 seconds" and a table with columns: Account, Buy/Sell, Symbol, Qty, Total cost, Transaction. The right pane is currently empty. A vertical scroll bar is visible between the two panes.

The panes on the left represent the MSGSenderSimulator application on servers Plants1 and Plants2. The panes on the right represent the Trade Processor application on servers Plants1 and Plants2.

- \_\_ b. Click **Send messages** for both servers Plants1 and Plants2.

**Sending messages from server: Plants1**

Select number of messages to be sent for each of the **Buy** and **Sell** categories, then click **Send messages**.

| Account                 | Buy/Sell | Symbol | Qty  | Total cost | Transaction |
|-------------------------|----------|--------|------|------------|-------------|
| 23423234 (John Doe)     | BUY      | PG     | 10.0 | \$657.90   | Plants1 - 5 |
| 87652289 (Elaine Moose) | SELL     | DELL   | 1.0  | \$19.89    | Plants1 - 6 |
| 23423234 (John Doe)     | BUY      | PG     | 10.0 | \$657.90   | Plants2 - 3 |
| 87652289 (Elaine Moose) | SELL     | DELL   | 1.0  | \$19.89    | Plants2 - 4 |

**Sending messages from server: Plants2**

Select number of messages to be sent for each of the **Buy** and **Sell** categories, then click **Send messages**.

| Account | Buy/Sell | Symbol | Qty | Total cost | Transaction |
|---------|----------|--------|-----|------------|-------------|
|         |          |        |     |            |             |

If the applications are running as configured, one server processes all of the messages. For the currently configured High Availability policy, only one server in the cluster is hosting the messaging engine.

- \_\_ c. You can test failover (of the messaging engine) by restarting the server that is currently processing the messages: Plants1 or Plants2. Send the messages again, and the other server, which is now hosting the messaging engine, processes them.

This last step concludes the verification of the service integration environment and the messaging applications.

## End of exercise

## Exercise review and wrap-up

In this exercise, you explored the structure and syntax of Ant build files and learned how to import property files into build files. Next, you used the ws\_ant utility to run the Ant build file to list, stop, update, and start an application. Finally, you ran several Ant tasks to configure a service integration environment.

# Reference

## ex10\_build.xml

```
<project name="ex10" default="init">

<!-- Read properties from file -->
<property resource="ex10.properties"/>

<taskdef name="wsadmin" classname="com.ibm.websphere.ant.tasks.WsAdmin"/>
<taskdef name="wsListApps"
classname="com.ibm.websphere.ant.tasks.ListApplications" />

<target name="init">

<!-- Create the time stamp -->
<tstamp/>

<echo message="--- Build of ${ant.project.name} started at ${TSTAMP} on
${TODAY} ---"/>
<echo message="--- Using ant version: ${ant.version} --- "/>
<echo message="ex10_build.xml"/>
</target>

<target name="stopApplication" description="Stops the specified
application.">
<echo message="--- Stopping the ${applicationName} application. ---"/>
<wsadmin script="${softwareDir}\ex10_stopApplication.py" lang="jython"
profile="${softwareDir}\ex10_profile.py" port="${soapPort}"
conntype="soap" user="${adminID}" password="${adminPasswd}">
<arg value="${applicationName}" />
<arg value="${clusterName}" />
</wsadmin>
</target>

<target name="startApplication" description="Starts the specified
application.">
<echo message="--- Starting the ${applicationName} applications. ---"/>
<wsadmin script="${softwareDir}\ex10_startApplication.py" lang="jython"
profile="${softwareDir}\ex10_profile.py" port="${soapPort}"
conntype="soap" user="${adminID}" password="${adminPasswd}">
<arg value="${applicationName}" />
<arg value="${clusterName}" />
</wsadmin>
</target>
```

```
<target name="updateApplicationAttribute" description="Updates a property  
in the specified application.">  
<echo message="---- Updating the ${applicationName} attribute. ---"/>  
<wsadmin script="${softwareDir}\ex10_updateApplicationAttribute.py"  
lang="jython" profile="${softwareDir}\ex10_profile.py" port="${soapPort}"  
connType="${connType}" user="${adminID}" password="${adminPasswd}">  
<arg value="${applicationName}" />  
<arg value="${attributeName}" />  
<arg value="${attributeValue}" />  
</wsadmin>  
</target>  
  
<target name="list" depends="init" description="Lists all applications">  
<wsListApps/>  
</target>  
</project>
```

## **Service integration environment build file (build.xml)**

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="Messaging" default="info">
<taskdef name="wsadmin" classname="com.ibm.websphere.ant.tasks.WsAdmin" />

<!--
All of the targets run scripts that take their arguments from the command
line. The arguments to pass to the scripts are retrieved from a properties
file.
-->
<property
file="/usr/Software/Messaging/Scripts/WDMConfiguration.properties" />

<target name="all" depends="
createUserAndAuthAlias,
createSIBus,
addUserToBusConnectorRole,
addClusterBusMember,
createSIBQueueDestination,
createSIBJMSConnectionFactory,
createSIBJMSQueue,
createSIBJMSActivationSpec,
installProducerApp,
installConsumerApp" />

<target name="info">
<echo>
Possible targets:
    createUserAndAuthAlias
createSIBus
addUserToBusConnectorRole
addClusterBusMember
createSIBQueueDestination
createSIBJMSConnectionFactory
createSIBJMSQueue
createSIBJMSActivationSpec
installProducerApp
installConsumerApp
</echo>
</target>

<!--
The createUserAndAuthAlias target is the only script that requires a
```

connection to the deployment manager. We must pass the username and password to the wsadmin command. The script creates a new user in the file based repository, and creates a JAAS authentication alias for the new user. The AuthAlias has the deployment manager's node name prepended to it.

-->

```
<target name="createUserAndAuthAlias">
<wsadmin conntype="SOAP" lang="jython" failonerror="true"
user="${adminuser}" password="${adminpassword}"
script="/usr/Software/Messaging/Scripts/createUserAndAuthAlias.py">
<arg value="${user}" />
<arg value="${password}" />
<arg value="${firstname}" />
<arg value="${surname}" />
<arg value="${alias}" />
</wsadmin>
</target>
```

<!--

The createSIBus creates an SIBus with the specified security setting  
-->

```
<target name="createSIBus">
<wsadmin conntype="NONE" lang="jython" failonerror="true"
script="/usr/Software/Messaging/Scripts/createSIBus.py">
<arg value="${bus}" />
<arg value="${busSecurity}" />
</wsadmin>
</target>
```

<!--

The addUserToBusConnectorRole target adds a user to the bus connector role  
-->

```
<target name="addUserToBusConnectorRole">
<wsadmin conntype="NONE" lang="jython" failonerror="true"
script="/usr/Software/Messaging/Scripts/addUserToBusConnectorRole.py">
<arg value="${bus}" />
<arg value="${user}" />
</wsadmin>
</target>
```

<!--

The addClusterBusMember target adds a cluster as a bus member to an SIBus. The bus member will be created using the specified policy, with a file store as its message store.

```
-->
<target name="addClusterBusMember">
<wsadmin conntype="NONE" lang="jython" failonerror="true"
script="/usr/Software/Messaging/Scripts/addClusterBusMember.py">
<arg value="${bus}" />
<arg value="${cluster}" />
<arg value="${fileStoreDirectory}" />
<arg value="${logDirectory}" />
<arg value="${policyName}" />
</wsadmin>
</target>

<!--
The createSIBQueueDestination target creates an SIBus Queue destination
associated with a cluster bus member.
-->
<target name="createSIBQueueDestination">
<wsadmin conntype="NONE" lang="jython" failonerror="true"
script="/usr/Software/Messaging/Scripts/createSIBQueueDestination.py">
<arg value="${bus}" />
<arg value="${sibDestinationName}" />
<arg value="${cluster}" />
</wsadmin>
</target>

<!--
The createSIBJMSConnectionFactory target creates a JMS Connection Factory
at cluster scope for the default messaging provider.
A container managed auth alias is configured for the connection factory.
-->
<target name="createSIBJMSConnectionFactory">
<wsadmin conntype="NONE" lang="jython" failonerror="true"
script="/usr/Software/Messaging/Scripts/createsIBJMSConnectionFactory.py">
<arg value="${bus}" />
<arg value="${jmsCFName}" />
<arg value="${jmsCFJndiName}" />
<arg value="${alias}" />
<arg value="${cluster}" />
</wsadmin>
</target>

<!--
The createSIBJMSQueue target creates a JMS Queue at cluster scope for the
default messaging provider. The JMS queue is associated with an SIBus
```

```

queue destination.

-->
<target name="createSIBJMSQueue">
<wsadmin conntype="NONE" lang="jython" failonerror="true"
script="/usr/Software/Messaging/Scripts/createSIBJMSQueue.py">
<arg value="${bus}" />
<arg value="${jmsQueueName}" />
<arg value="${jmsQueueJndiName}" />
<arg value="${sibDestinationName}" />
<arg value="${cluster}" />
</wsadmin>
</target>

<!--
The createSIBJMSActivationSpec target creates a JMA Activation Spec at
cluster scope for the default messaging provider. The Activation Spec
is associated with a JMS destination. The Activation Spec will connect
to the SIBus using the provided auth alias.

-->
<target name="createSIBJMSActivationSpec">
<wsadmin conntype="NONE" lang="jython" failonerror="true"
script="/usr/Software/Messaging/Scripts/createSIBJMSActivationSpec.py">
<arg value="${bus}" />
<arg value="${jmsASName}" />
<arg value="${jmsASJndiName}" />
<arg value="${jmsQueueJndiName}" />
<arg value="${alias}" />
<arg value="${cluster}" />
</wsadmin>
</target>

<!--
The installProducerapp target installs the JMS producer app to a cluster.
It associates the JMS queue with the destination reference in the app,
and associates the JMS Connection Factory with the resource ref in the app,
setting the auth alias used to connect to the connection factory.

-->
<target name="installProducerApp">
<wsadmin conntype="NONE" lang="jython" failonerror="true"
script="/usr/Software/Messaging/Scripts/installProducerApp.py">
<arg value="${producerapp}" />
<arg value="${jmsQueueJndiName}" />
<arg value="${jmsCFJndiName}" />
<arg value="${alias}" />

```

```
<arg value="${cluster}" />
</wsadmin>
</target>

<!--
# This script installs the JMS consumer app to a cluster with all other
install options at default values.
-->
<target name="installConsumerApp">
<wsadmin conntype="NONE" lang="jython" failonerror="true"
script="/usr/Software/Messaging/Scripts/installConsumerApp.py">
<arg value="${consumerapp}" />
<arg value="${cluster}" />
</wsadmin>
</target>
</project>
```

# Exercise 11. Automating the installation of WebSphere Application Server

## What this exercise is about

In this exercise, you learn how to automate the installation of WebSphere Application Server and the creation of profiles by doing silent installations.

## What you should be able to do

At the end of this exercise, you should be able to:

- Describe the silent installation of IBM Installation Manager
- Use the IBM Installation Manager to record a response file
- Use a recorded response file to silently install the WebSphere Application Server Network Deployment product files
- Edit the response file to modify the installation
- Customize a response file to create a WebSphere profile by using the manageprofiles command

## Introduction

WebSphere Application Server provides a mechanism to install the product without using a graphical user interface: silent installation. By using a file to supply installation options, you can install WebSphere Application Server without user interaction. This procedure reduces the chance of human error, ensures consistency, and promotes reuse. For example, after you specify the wanted options, you can use the same response file to install WebSphere the same way on multiple machines.

Silent installation uses the same IBM Installation Manager as in the graphical installation. However, instead of displaying a graphical user interface to prompt for installation options, the installation manager reads your responses from a file that you provide. You can also use the IBM Installation Manager to record a response file during a graphical installation.

## Requirements

To complete this exercise, you need the WebSphere Application Server Network Deployment V8.5 product files that are stored in a local repository **/usr/IBM-repositories/WAS85**. The IBM Installation Manager must also be installed.

## Exercise instructions

### Section 1: Read-only: Silent installation of IBM Installation Manager (IIM)

This section is for reference since the IBM Installation Manager is already installed on the lab image.

First, of all, you should understand the difference between these two items:

- **IBM Installation Manager Kit:** The kit provides a directory that contains an instance of IBM Installation Manager that installs a package along with IBM Installation Manager. The file that starts Installation Manager is named `install`. You can configure the kit to install both Installation Manager and one or more packages.
- **IBM Installation Manager:** This item is the installed version of IBM Installation Manager. The name of the file that starts Installation Manager is `IBMMIM`, and it is installed by default into the `/opt/IBM/Installation Manager` directory.

You must use the IBM Installation Manager Kit to install IBM Installation Manager on a client computer.

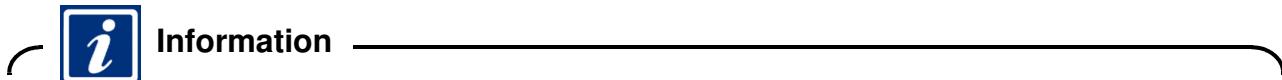
- \_\_\_ 1. Download the IBM Installation Manager Kit, `agent.installer.platform.zip`, for a specific platform. For example, a recent version for Linux is named `agent.installer.linux.gtk.x86_1.5.3000.20120531.zip`.
- \_\_\_ 2. Extract the compressed file to a directory.
- \_\_\_ 3. Install by using the Installation Manager silent installation command
  - \_\_\_ a. Run `./installc -acceptLicense -log log_file_path_and_name`
- \_\_\_ 4. Add a repository and download product packages to it. A **repository** is where the installable packages are found. The repository includes metadata that describes the software version and how it is installed. It has a list of files that are organized in a tree structure. The repository can be local or on a remote server. A **package** is a software product that Installation Manager installs. It is a separately installable unit that can operate independently from other packages of that software.

## Section 2: Silent installation of WebSphere Application Server Network Deployment core product files

In this section, you learn how to use the IBM Installation Manager to record a response file. You then use a response file to install the WebSphere Application Server product files.

WebSphere Application Server is already installed on your lab image, so you create another instance on the same host in this exercise for illustrative purposes.

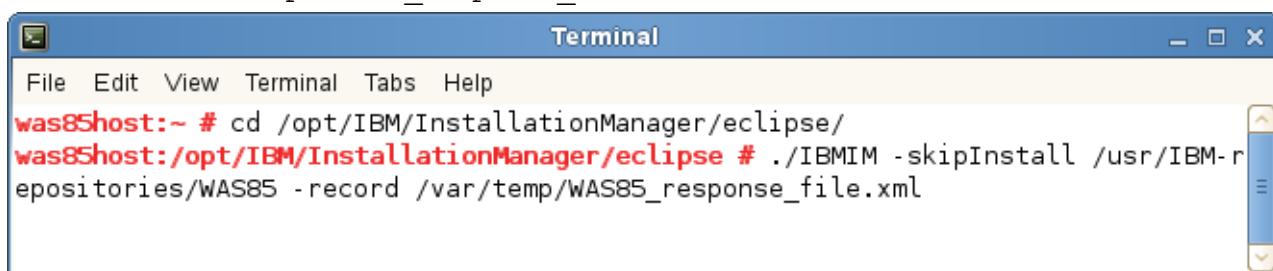
- 1. Make sure that all WebSphere servers are shut down: application servers, node agents, and the deployment manager.
- 2. The next information block is for reference only. Skip to the next step that follows this information block to proceed with the exercise.



### Using the IBM Installation Manager to record a response file

Here are the steps for recording the response file for installing the WebSphere Application Server ND core product files. This information is for reference only. For this exercise, you are going to use an existing response file that is provided for you.

1. Start IIM in record mode.
  - a. Open a Terminal window, and go to the directory  
`/opt/IBM/InstallationManager/eclipse`
  - b. Enter the following command:  
`./IBMMIM -skipInstall /usr/IBM-repositories/WAS85 -record /var/temp/WAS85_response_file.xml`

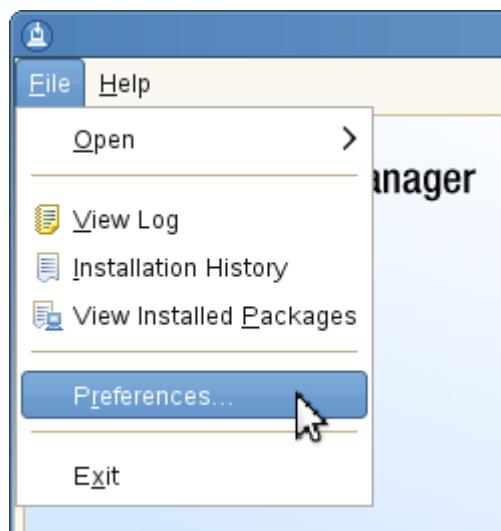


A screenshot of a terminal window titled "Terminal". The window has a blue header bar with the title. Below the title is a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area of the terminal shows a command-line session. The user is in a directory called "was85host:~". They type the command `cd /opt/IBM/InstallationManager/eclipse/` and press enter. Then they type the command `./IBMMIM -skipInstall /usr/IBM-repositories/WAS85 -record /var/temp/WAS85_response_file.xml` and press enter again. The terminal window has a scroll bar on the right side.

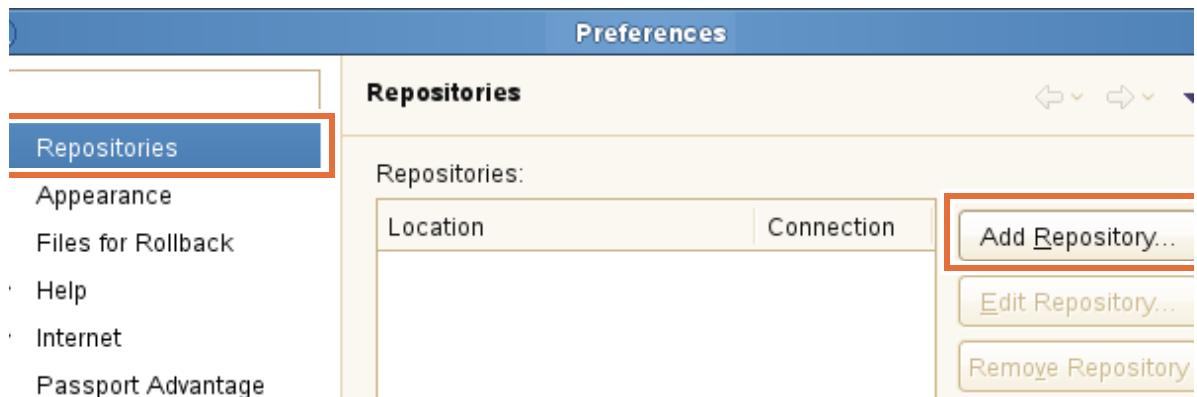
```
was85host:~ # cd /opt/IBM/InstallationManager/eclipse/
was85host:/opt/IBM/InstallationManager/eclipse # ./IBMMIM -skipInstall /usr/IBM-repositories/WAS85 -record /var/temp/WAS85_response_file.xml
```

2. Add a repository.

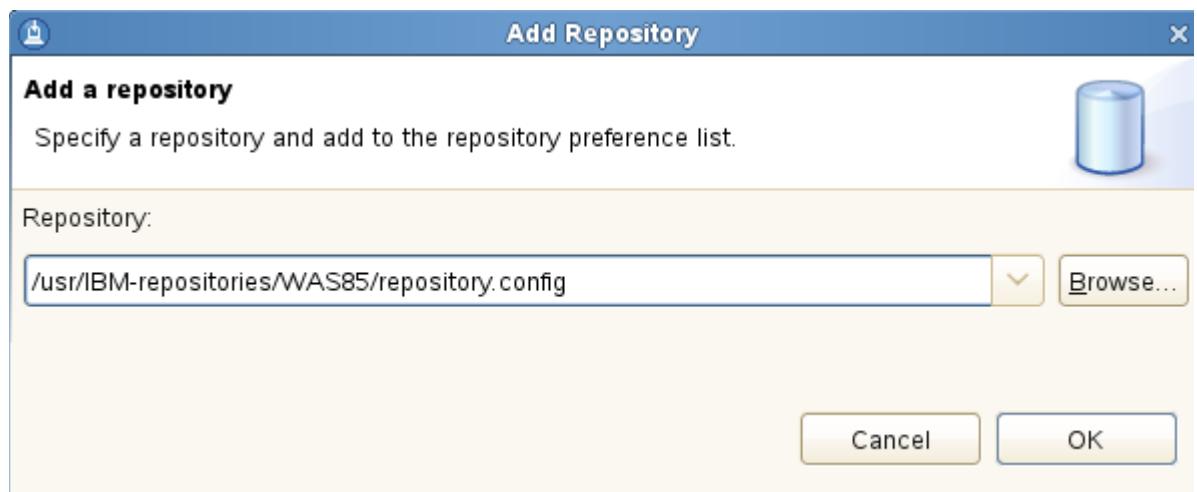
a. When the IIM GUI starts, click **File > Preferences**.



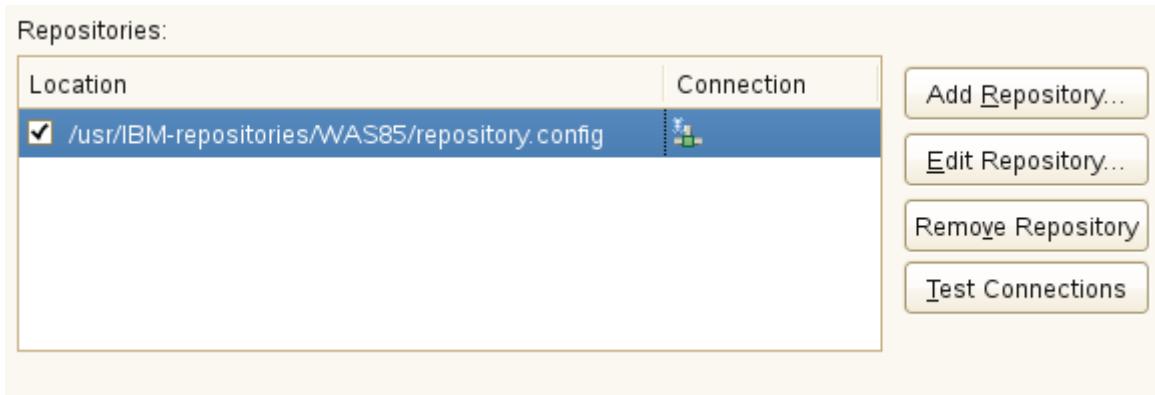
b. Select Repositories and click **Add a repository**.



c. Browse to /usr/IBM-repositories/WAS85/ and select repository.config



d. Click **OK**.

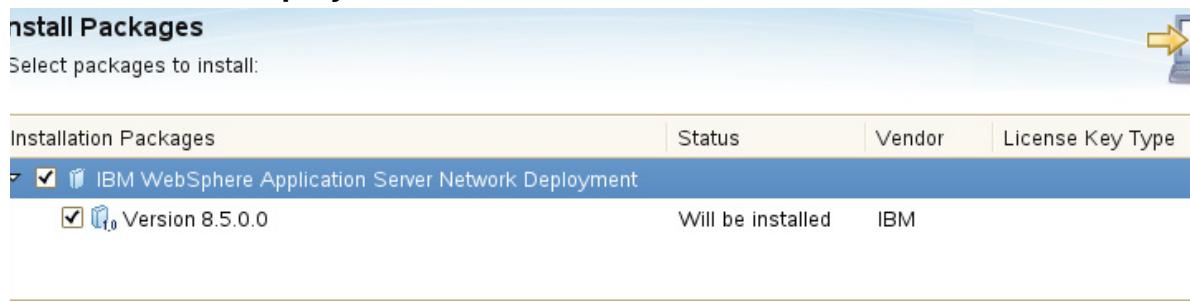


- e. You see the repository location listed. Click **Test Connections**. When the connection succeeds, click **OK**. Click **OK** again.
3. Use IIM to record a response file.

a. Click **Install** on the IIM screen.



- b. On the Install Packages screen, check **IBM WebSphere Application Server Network Deployment**, and click **Next**.



- c. If the installation package is already installed, see a warning screen that shows that the package is already installed.



- d. You can click **Continue** to install another instance of the package.  
e. Select **I accept the terms in the license agreement**, and click **Next**.  
f. Keep the default location for the Shared Resources Directory, and click **Next**.

Shared Resources Directory:	/opt/IBM/IMShared	<a href="#">Browse...</a>
<b>Disk Space Information</b>		
Volume	Available Space	
/	16.90 GB	

- g. On the next screen, make sure that **Create a new package group** is selected.

▷ [Create a new package group](#)

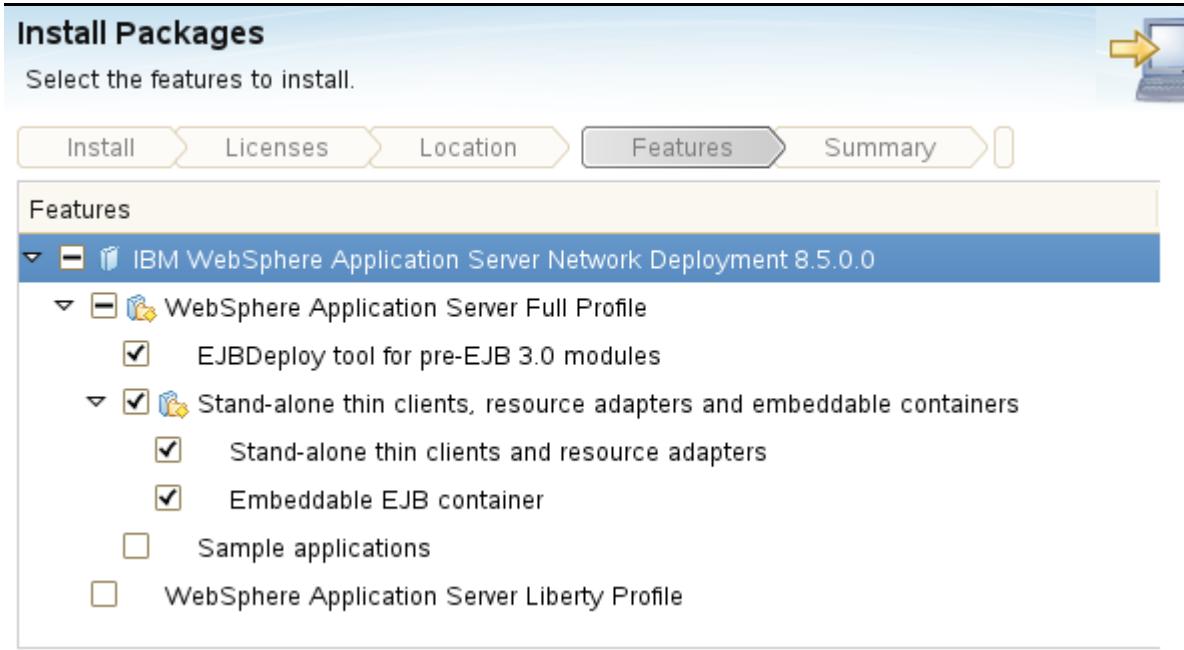
Package Group Name	Installation Directory
IBM WebSphere Application Server V8.5_1	/opt/IBM/WebSphere/AppServer_1

Package Group Name: IBM WebSphere Application Server V8.5\_1

Installation Directory: /opt/IBM/WebSphere/AppServer\_1

- h. If you are installing another instance on the same host, the installation manager makes the Package Group Name and Installation Directory unique name by appending \_1.

- i. Click **Next**.
- j. On the next screen, make sure that English is selected, and click **Next**.
- k. The Features panel lists more features for installation. Note the default selections and click **Next**.



- l. On the Summary screen, click **Install**.
- m. The “installation” takes a brief amount of time, and the product is not installed since you used the `-skipInstall` command-line option. However, as you are going to see, the installation choices are recorded in a response file.
- n. The response file that is generated is written to the file you specified on the command line after the `-record` options. In this example, `-record /var/temp/WAS85_response_file.xml`
- o. Click **Finish**.
- p. Click **File > Exit** to exit the IBM Installation Manager.

4. Examine the response file that you are going to use to install the product.
- a. From a terminal window, go to `/usr/Software/Scripts/Exercise11`.

- \_\_ b. Open **WAS85\_response\_file.xml** by using a text editor such as gedit.

```
Terminal
File Edit View Terminal Tabs Help
was85host:~ # cd /usr/Software/Scripts/Exercise11
was85host:/usr/Software/Scripts/Exercise11 # ls
WAS85_response_file.xml
was85host:/usr/Software/Scripts/Exercise11 # gedit WAS85_response_file.xml
```

- \_\_ c. Examine the top sections of the response file.

```
<?xml version="1.0" encoding="UTF-8"?>
<! --The "acceptLicense" attribute has been deprecated.-->
<agent-input acceptLicense='true'>
<server>
<repository location='/usr/IBM-repositories/WAS85' />
</server>
<profile id='IBM WebSphere Application Server V8.5_2'
  installLocation='/opt/IBM/WebSphere/AppServer_2'>
<data key='eclipseLocation' value='/opt/IBM/websphere/AppServer_2' />
<data key='user.import.profile' value='false' />
<data key='cic.selector.os' value='linux' />
<data key='cic.selector.ws' value='gtk' />
<data key='cic.selector.arch' value='x86' />
<data key='cic.selector.nl' value='en' />
</profile>
```

**acceptLicence = 'true'** The “acceptLicense attribute in the response file is deprecated. Use “-acceptLicense” command-line option to accept license agreements.

**profile is=' IBM WebSphere Application Server V8.5\_2'** Since you use this response file to create another instance on the same host, the profile name is made unique by appending the \_2.

**installLocation=' /opt/IBM/WebSphere/AppServer\_2'** Again, Since you use this response file to create another instance on the same host, the installation location is made unique.

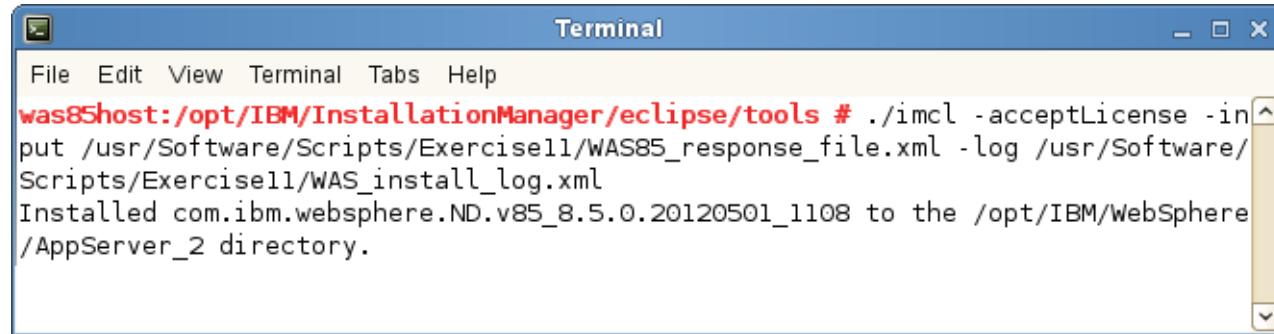
- \_\_ d. Notice the following lines.

```
<install modify='false'>
<offering id='com.ibm.websphere.ND.v85' version='8.5.0.20120501_1108'
  profile='IBM WebSphere Application Server V8.5_2'
  features='core.feature,ejbdeploy,thinclient,embeddablecontainer,
  com.ibm.sdk.6_32bit' installFixes='none' />
</install>
```

The features list contains the names of features that are installed by default.

- **core.feature** indicates the full WebSphere Application Server profile
- **ejbdeploy** indicates the EJBDeploy tool for pre-EJB 3.0 modules
- **thinclient** indicates the standalone thin clients and resource adapters
- **embeddablecontainer** indicates the embeddable EJB container
- **com.ibm.sdk.6\_32bit** allows you to choose a 32-bit Software Development Kit if you are installing on a 64-bit system
- **samples** indicates the sample applications feature (This feature is not listed in the response file because it is not installed by default. You learn how to modify an existing installation by adding this feature later in the exercise.)

- \_\_\_ e. Close the **WAS85\_response\_file.xml** file.
- \_\_\_ 5. Run Installation Manager command line (imcl) installation command.
  - \_\_\_ a. From the Terminal window, go to  
**/opt/IBM/InstallationManager/eclipse/tools**
  - \_\_\_ b. Enter the following command all on one line:  
`./imcl -acceptLicense -input /usr/Software/Scripts/Exercise11/WAS85_response_file.xml -log /usr/Software/Scripts/Exercise11/WAS_install_log.xml`
  - \_\_\_ c. Wait a few minutes for the installation to complete. Monitor the processor usage.

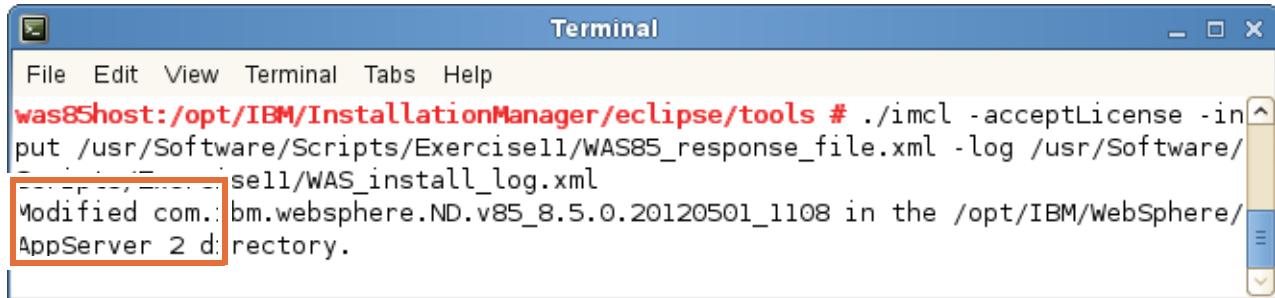


```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/InstallationManager/eclipse/tools # ./imcl -acceptLicense -input /usr/Software/Scripts/Exercise11/WAS85_response_file.xml -log /usr/Software/Scripts/Exercise11/WAS_install_log.xml
Installed com.ibm.websphere.ND.v85_8.5.0.20120501_1108 to the /opt/IBM/WebSphere/AppServer_2 directory.
```

- \_\_\_ 6. -Examine file system structure.
  - \_\_\_ a. Use a file system browser such as Nautilus to go to the new installation directory **/opt/IBM/WebSphere/AppServer\_2**.
  - \_\_\_ b. Notice that there is no samples folder because the samples feature was not included on the features line of the response file.
- \_\_\_ 7. Modify the installation.
  - \_\_\_ a. From a terminal window, go to **/usr/Software/Scripts/Exercise11**.
  - \_\_\_ b. Open the **WAS85\_response\_file.xml** file in a text editor such as gedit.
  - \_\_\_ c. Make the following change. Add **samples** to the features list as shown,

```
features='core.feature,ejbdeploy,thinclient,embeddablecontainer,com.ibm.sdk.6_32bit,samples' installFixes='none' />
```

- \_\_\_ d. Save the changes and close the `WAS85_response_file.xml` file.
- \_\_\_ e. Run the `imcl` command again.



```
Terminal
File Edit View Terminal Tabs Help
was85host:/opt/IBM/InstallationManager/eclipse/tools # ./imcl -acceptLicense -input /usr/Software/Scripts/Exercisell/WAS85_response_file.xml -log /usr/Software/Exercisell/WAS_install_log.xml
Modified com.ibm.websphere.ND.v85_8.5.0.20120501_1108 in the /opt/IBM/WebSphere/AppServer_2 directory.
```

- \_\_\_ f. Notice that the message now indicates that the instance of `com.ibm.websphere.ND.v85` was Modified.
- \_\_\_ g. Go to the directory `/opt/IBM/WebSphere/AppServer_2` and verify that there is now a `samples` folder.

### **Section 3: Create profile1 by using a response file and the manageprofiles command**

In this section, you create the response file that you can use with the manageprofiles command to create a profile.

- \_\_\_ 1. Develop a response file for creating a stand-alone profile.

- The value of <was\_root> is /opt/IBM/WebSphere/AppServer\_2
- The value of <profile\_root> is  
/opt/IBM/WebSphere/AppServer\_2/profiles

**Table 8: Argument values for profile1 profile creation**

	<b>Argument name</b>	<b>Argument value</b>
1	templatePath	<was_root>/profileTemplates/managed
2	profileName	profile1
3	profilePath	<profile_root>/profile1
4	enableAdminSecurity	true
5	adminUserName	wasadmin
6	adminPassword	websphere

- \_\_\_ a. The response file can include the following lines.

```
create

templatePath=/opt/IBM/WebSphere/AppServer_2/profileTemplates/
default

profileName=profile1

profilePath=/opt/IBM/WebSphere/AppServer_2/profiles/profile1

enableAdminSecurity=true

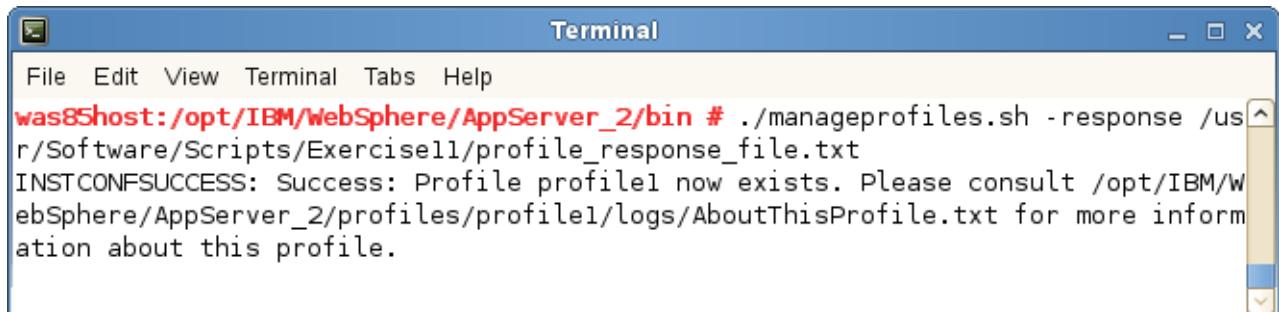
adminUserName=wasadmin

adminPassword=websphere
```

- \_\_\_ b. Use a text editor to create a file named `profile_response_file.txt` that contains these lines and save it to `/usr/Software/Scripts/Exercise11`

- \_\_\_ c. From a Terminal window, go to `/opt/IBM/WebSphere/AppServer_2/bin`

- \_\_\_ d. Enter the following command: `./manageprofiles.sh -response /usr/Software/Scripts/Exercise11/profile_response_file.txt`



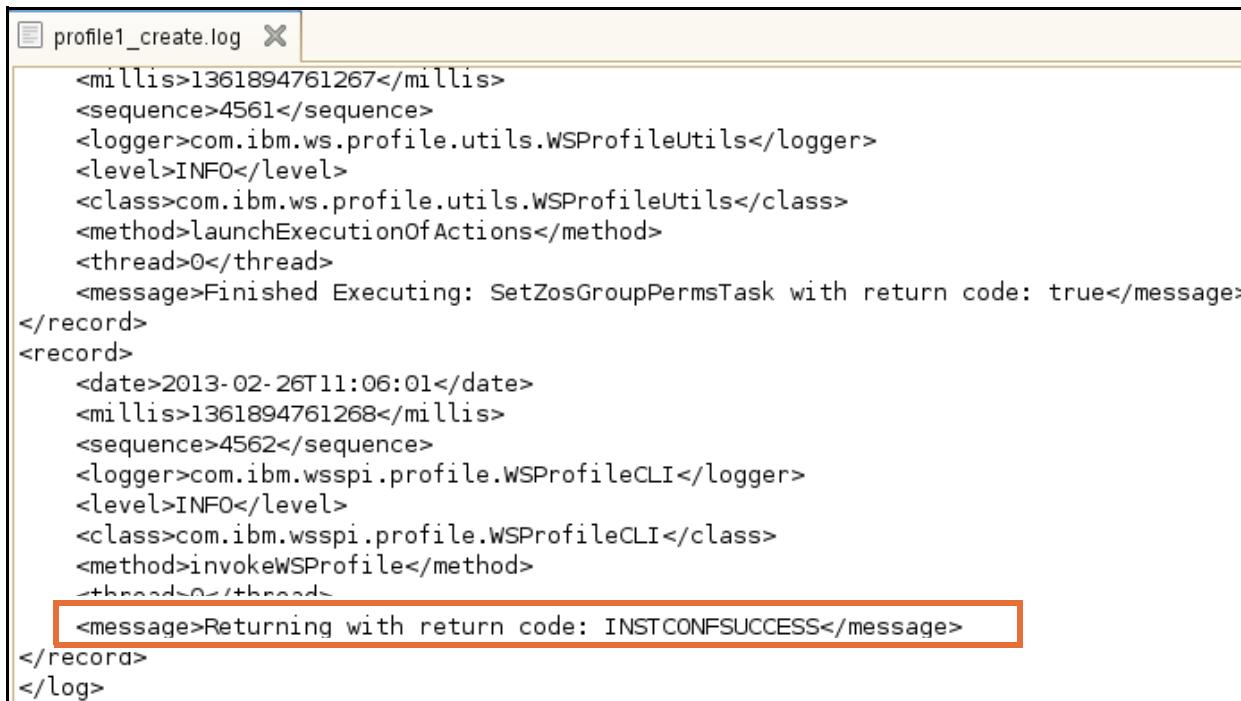
The screenshot shows a terminal window with the title "Terminal". The menu bar includes "File", "Edit", "View", "Terminal", "Tabs", and "Help". The command entered was `./manageprofiles.sh -response /usr/Software/Scripts/Exercise11/profile_response_file.txt`. The terminal output is:  
**was85host:/opt/IBM/WebSphere/AppServer\_2/bin #** `./manageprofiles.sh -response /usr/Software/Scripts/Exercise11/profile_response_file.txt`  
INSTCONFSUCCESS: Success: Profile profile1 now exists. Please consult /opt/IBM/WebSphere/AppServer\_2/profiles/profile1/logs/AboutThisProfile.txt for more information about this profile.

- \_\_\_ e. If you do not see the INSTCONFSUCCESS message, correct any errors in the response file, and run the `manageprofiles` command again.

## Section 4: Verify installation WebSphere Application Server and profile1

In this section, you check the create profile log to verify that profile1 was successfully created, and examine the file system structure for profile1. Then, you start the application server, server1, defined in profile. Finally, you access the administrative console and the snoop servlet to verify that those applications are running properly.

- \_\_\_ 1. Examine the manageprofiles logs.
  - \_\_\_ a. Use a file system browser such as Nautilus to go to  
**/opt/IBM/WebSphere/AppServer\_2/logs/manageprofiles**
  - \_\_\_ b. Open the file `profile1_create.log` with a text editor.
  - \_\_\_ c. Scroll to the bottom of the file, and confirm that you see a line with the following message: INSTCONFSUCCESS. This message confirms that the profile creation completed successfully.



```

profile1_create.log X
<millis>1361894761267</millis>
<sequence>4561</sequence>
<logger>com.ibm.ws.profile.utils.WSProfileUtils</logger>
<level>INFO</level>
<class>com.ibm.ws.profile.utils.WSProfileUtils</class>
<method>launchExecutionOfActions</method>
<thread>0</thread>
<message>Finished Executing: SetZosGroupPermsTask with return code: true</message>
</record>
<record>
  <date>2013-02-26T11:06:01</date>
  <millis>1361894761268</millis>
  <sequence>4562</sequence>
  <logger>com.ibm.wsspi.profile.WSProfileCLI</logger>
  <level>INFO</level>
  <class>com.ibm.wsspi.profile.WSProfileCLI</class>
  <method>invokeWSProfile</method>
  <thread>0</thread>
  <message>Returning with return code: INSTCONFSUCCESS</message>
</record>
</log>

```

 **Troubleshooting**

The `Profile_name_create.log` file contains trace messages for all events that occur during the creation of the named profile. It shows the final status of the process by using one of the following return codes:

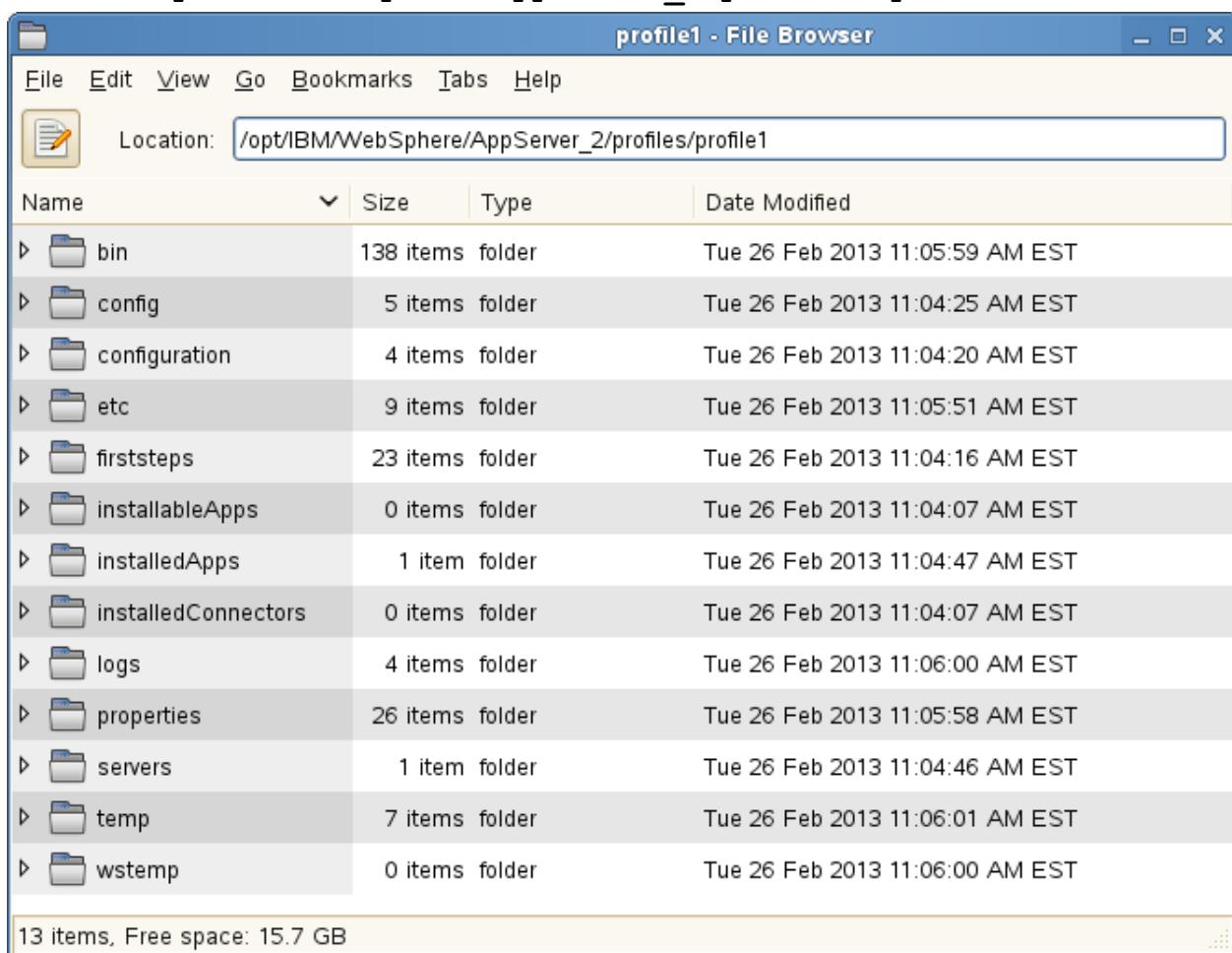
- `INSTCONFFAIL` = Total profile creation failure
- `INSTCONFSUCCESS` = Successful profile creation
- `INSTCONFPARTIALSUCCESS` = Profile creation errors occurred but the profile is still functional. Other information identifies the errors.

If you do not get an INSTCONFSUCCESS return code, look in the log for any message that might give you an indication of the error. To recover:

1. Review your response file to make sure that you specified the wanted options and values correctly.
2. If the return code is INSTCONFPARTIALSUCCESS, delete the partially created profile by issuing the following commands in the Terminal window:
 

```
cd /opt/IBM/WebSphere/AppServer_2/bin
./manageprofiles.sh -delete -profileName profile1
```
3. If the profile root folder exists (/opt/IBM/WebSphere\_2/profiles/profile1), delete it.
4. Restart the silent installation.

- 5. Examine the file system structure for profile1. There is now a profile that is listed in the profiles directory.
- a. Use a file system browser such as Nautilus to go to  
**/opt/IBM/WebSphere/AppServer\_2/profiles/profile1**



- \_\_\_ b. Expand the **logs** folder and right-click the file **AboutThisProfile.txt**. Open the file with gedit.

```
Application server environment to create: Application server
Location: /opt/IBM/WebSphere/AppServer_2/profiles/profile1
Disk space required: 200 MB
Profile name: profile1
Make this profile the default: True
Node name: was85hostNode01
Host name: was85host
Enable administrative security (recommended): True
Administrative console port: 9063
Administrative console secure port: 9046
HTTP transport port: 9082
HTTPS transport port: 9445
Bootstrap port: 2811
SOAP connector port: 8883
Run application server as a service: False
Create a Web server definition: False
Performance tuning setting: Standard
```

- \_\_\_ c. It is important to take note of the ports that profile1 uses. Yours might be different from those values that are shown in the screen capture. Enter your actual port values.

Administrative console port: \_\_\_\_\_

HTTP transport port: \_\_\_\_\_

- \_\_\_ d. Close the **AboutThisProfile.txt** file.

- \_\_\_ 6. Start server1.

- \_\_\_ a. From a Terminal window, go to

**/opt/IBM/WebSphere/AppServer\_2/profiles/profile1/bin**

- \_\_\_ b. Enter the command: **./startServer.sh server1**

```
File Edit View Terminal Tabs Help
was85host:/opt/IBM/WebSphere/AppServer_2/bin # ./startServer.sh server1
ADMU0116I: Tool information is being logged in file
          /opt/IBM/WebSphere/AppServer_2/profiles/profile1/logs/server1/startSe
rver.log
ADMU0128I: Starting tool with the profile1 profile
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server server1 open for e-business; process id is 29850
```

- \_\_\_ c. Wait for the server to start.
- \_\_\_ 7. Access the administrative console.
- \_\_\_ a. Open a web browser and enter the following web address. Make sure to use your administrative console port if it is different from 9063.

<http://was85host:9063.ibm/console>



## This Connection is Untrusted

You have asked Firefox to connect securely to **was85host:9046**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

### What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

[Get me out of here!](#)

#### ► Technical Details

#### ▼ I Understand the Risks

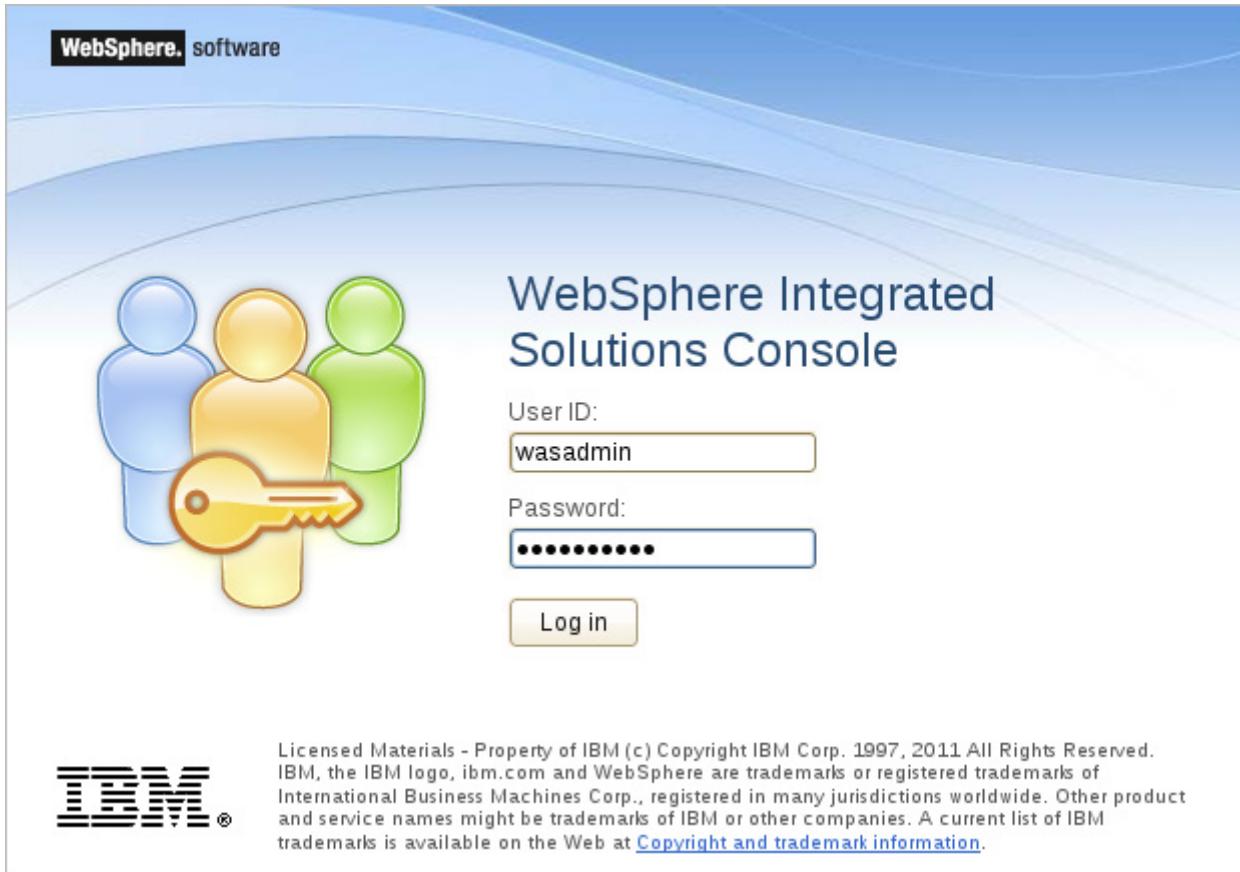
If you understand what's going on, you can tell Firefox to start trusting this site's identification. **Even if you trust the site, this error could mean that someone is tampering with your connection.**

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

[Add Exception...](#)

- \_\_\_ b. Expand **I Understand the Risks**, and click **Add Exception**.

- \_\_ c. On the Add Security Exception screen, click **Confirm Security Exception**.



- \_\_ d. Log in to the WebSphere Integrated Solutions Console by using the **User ID** wasadmin and **Password** web1sphere.
- \_\_ e. Explore the stand-alone environment of this new server to verify that the console is working properly.
- \_\_ f. Log out of the console.
- \_\_ 8. Access the snoop servlet.
- \_\_ a. Open a web browser and enter the following web address. Make sure to use your HTTP transport port if it is different from 9082.

http://was85host:9082/snoop

The screenshot shows a Mozilla Firefox window titled "Snoop Servlet - Mozilla Firefox". The address bar displays "http://was85host:9082/snoop". The main content area has a title "Snoop Servlet - Request/Client Information". It includes sections for "Requested URL:" (containing "http://was85host:9082/snoop"), "Servlet Name:" (containing "Snoop Servlet"), and "Request Information:" (containing a table with two rows: "Request method" and "GET" in the first row, and "Request URI" and "/snoop" in the second row).

Request method	GET
Request URI	/snoop

This step concludes the installation and profile creation verification process.

**End of exercise**

## Exercise review and wrap-up

In this exercise, you performed a silent installation of WebSphere Application Server V8.5 Network Deployment. You first familiarized yourself with the structure and content the sample options response file. You then customized it to specify the options that are required for a product code only installation and used it in a silent invocation of the IBM Installation Manager. Then, you customized a response file again and used it to perform a silent installation to create a stand-alone profile. Finally, you verified and confirmed that the server environment and its applications were operational.



**IBM**  
®