Tivoli Intelligent Orchestrator

Using the Automation Package Development Environment

August 3, 2004

IBM Tivoli Intelligent Orchestrator is an automated resource management solution for corporate and Internet data centers. Tivoli Intelligent Orchestrator configures resources among applications in a multi-application environment to balance end-user traffic demands, excess capacity, and service level targets.

Tivoli Intelligent Orchestrator uses historical and current demand and performance data, as well as defined business policies and service level objectives to determine where and when to allocate resources. Once a deployment decision is made, Tivoli Intelligent Orchestrator runs workflows to automatically make the required configuration and allocation changes.

Workflows automate processes from configuring and allocating servers, to installing, configuring, and patching software. For example, modifications to data center infrastructure (route changes, VLAN assignments), configuration and allocation of servers (software installation, configuration), and specific command actions (reboot server, power off device, install image) can all be accomplished using workflows.

Workflow components

Logical operations define an action that should be implemented by an automation package for a particular resource of that type.

For example, **IPSystem.AddNetworkInterface** is a logical operation that adds an IP address to a device. It is a logical operation in that it makes no assumptions about the operating environment that will contain the new IP address. For example, you may want to add an IP address to a Linux environment or to Windows environment, and you can accomplish this task starting with the same logical operation.

From there, you will run a specific workflow for you Linux or Windows environment that will actually add the IP address.

A *device driver*, also referred to as a device model, represents different manufacturers' products and versions and is identified by its own make and model number. For example, all Cisco CSS 11050 switches in a data center can perform the same action using the same workflow. This workflow, in turn, is dependent on the make and model of the switch and not a specific instance of it. Each data center device can either inherit all of its workflows from the associated device driver or can override one or more with its own custom workflows.

Workflow composer

Full editing capabilities for workflow design are provided in the workflow composer. The workflow composer is a powerful toolset that is designed to allow you to create and maintain workflows in Tivoli Intelligent Orchestrator. The workflow composer supports standard programming elements: parameters, variables, assignments, expressions, loops, iterators, conditionals, comments, try, catch, catch all, finally and throw as well as the following elements:

Element	Description
Scriptlet	A scriptlet represents a list of commands that can be run without user interaction. The workflow composer supports scripts written in Bash, Perl, and Expect.
Check Device Locale	Check the language (locale) for the device. If a locale has been specified, a workflow will fail if the target device for the workflow does not match the locale.
DCM Insert	A data center model insert statement represents a call to the data center model to insert new information into the database.
DCM Update	A data center model update statement represents a call to the data center model to update new information into the database.
DCM Delete	A data center model delete statement represents a call to the data center model to delete information from the database.
Log	A log element logs details associated with the workflow when it is run. You can choose to log messages of type: Info, Warning, or Error.

For more information on these elements, refer to the online help.

Installing the Automation Package Development Environment

Prerequisites:

1. Install IBM J2RE 1.4.1, if you have not already done so. If you have WSAD installed, the JRE is located here: <*WSAD_HOME*> /runtimes/base_v51/java/.
2. If you are installing Eclipse on the same machine as Tivoli Intelligent Orchestrator and WebSphere Application Server, then the Java Development Kit (JDK) is available in (WAS _Install_directory)/java.
3. Your JRE executables must be defined as a PATH environment variable or Eclipse will not start. If you have more than one JRE installed, make sure the IBM JRE is the first one in your path: PATH=%PATH%;C:\IBM\WebSphere\AppServer\java\jre\bin
4. The Automation Package Development Environment has only been tested on the Windows platform.

To install and configure the Automation Package Development Environment:

1. Set the JAVA_HOME variable to your IBM JRE installation directory. For example, /IBM/WebSphere/AppServer/java. The Automation Package Development Environment will only work with the IBM JRE.
2. Install the latest version of Eclipse. The latest version is available at: http://www.eclipse.org/downloads/index.php. You will need to download the

Eclipse SDK, and not just the platform runtime. Select the appropriate Eclipse build for your platform.

3. Extract the eclipse zip file to a directory called **APDE_HOME**.
4. Point your browser to: http://www-18.lotus.com/wps/portal/automation/.
5. In the Search field, type **Automation Package Development Environment**.
6. Download the **com.ibm.tivoli.orchestrator.tcdriverdevelopment.zip** file.
7. Extract the **com.ibm.tivoli.orchestrator.tcdriverdevelopment.zip** file to the **APDE_HOME/eclipse** directory
8. Navigate to the **APDE_HOME /eclipse** directory and double-click the **eclipse.exe** file to start Eclipse.

Configuring the Automation Package Development Environment

To compile and run workflows from the Automation Package Development Environment, you need to configure the location of your configuration files and specify the name of the Tivoli Intelligent Orchestrator server.

To configure the Automation Package Development Environment:

1. From the Eclipse menu, select **Window** -> **Preferences**.
2. Select **Automation Package**.
3. In the **Server Name** field, type the name or IP address of the Tivoli Intelligent server that you will use to compile and run your workflows.
4. In the **Directory** field, specify the directory that will contain your configuration files.
5. Before you can start the Automation Package Development Environment, copy the **dcm.xml** file from the <TIO_HOME>\config directory of your Tivoli Intelligent Orchestrator server installation to the directory that you specified in Step 4. Alternatively, if the Automation Package Development Environment is installed on the same machine as your Tivoli Intelligent Orchestrator server, click **Window** > **Preferences** > **Automation Package** and type **\IBM\tivoli\thinkcontrol\config** in the **Directory** field.
6. If you have not already configured your database, modify dcm.xml by replacing **YourDBUserName** and **YourDBPassword** with the correct values. Contact your database administrator for the URL specified in the **dcm.xml** file if you do not have it. You may need to install the database client if the database you are using is not installed on the same machine as the Automation Package Development Environment. The database that is being used should be the same database as the one used by Tivoli Intelligent Orchestrator.
7. Ensure that the database being used is correctly defined in dcm.xml.

Setting preferences

To set for the preferences for the workflow composer:

1. Click **Window** and select **Preferences**. The Preferences window is displayed.
2. In the Preferences window, select **Automation Package**.

3. Set your preferences on the **Automation Package** window.
4. Click **Workflow Editor Preference** to set your preferences for the workflow composer.

Using the Automation Package Development Environment

Make sure you are in the Automation Package perspective when developing automation packages. To switch perspectives, click **Window > Open Perspective** and select **Automation Package**.

Creating a new automation package

To create a new automation package in the Automation Package Development Environment:

1. From the Eclipse menu, select **File** > **New** > **Other**.
2. Expand **Automation Package Development** and select **Automation Package Project**.
3. Type the new project name on the New Automation Package Project window.
4. Click **Finish**. The new project is created in the Navigator view.

Using the workflow composer

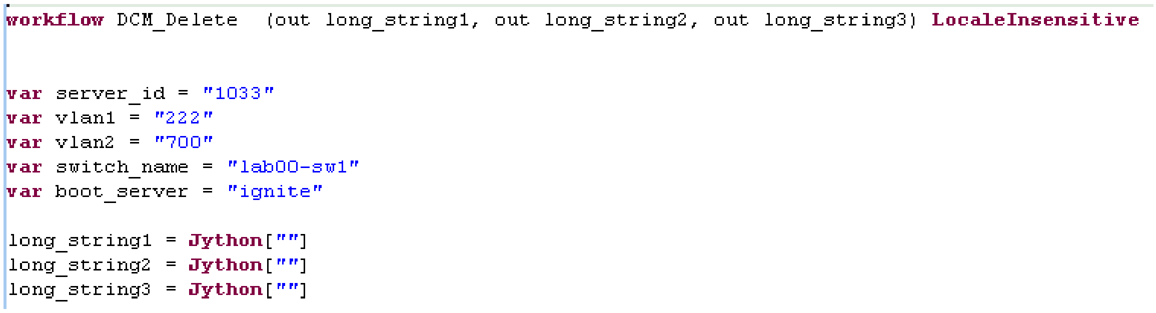
Once you have created a new Automation Package project, you can write new workflows using the workflow composer. The workflow composer is used to create and edit workflow (.wkf) files that contain the information that is needed to operate a specific physical device.

In the workflow composer, type a name for the workflow with the following naming convention: workflow <workflow name>, For example:

```
workflow DCM_Delete  (out long_string1, out long_string2, out
long_string3) LocaleInsensitive
```

Type LocaleInsensitive if you are not concerned with a particular locale. If a locale has been specified, a workflow will fail if the target device for the workflow does not match the locale.

You must also specify any output variables that you need to declare for your workflow. In the example below, `long_string1`, `long_string2`, and `long_string3` represent output variables.



```
workflow DCM_Delete  (out long_string1, out long_string2, out long_string3) LocaleInsensitive


var server_id = "1033"
var vlan1 = "222"
var vlan2 = "700"
var switch_name = "lab00-sw1"
var boot_server = "ignite"

long_string1 = Jython[""]
long_string2 = Jython[""]
long_string3 = Jython[""]
```

The workflow composer supports the Jython programming language and you should follow common programming techniques when creating a new workflow. It also supports if...then , if...then...else, foreach, and while statements conditional statements and try...catch...catchall...finally and throw error handling techniques.

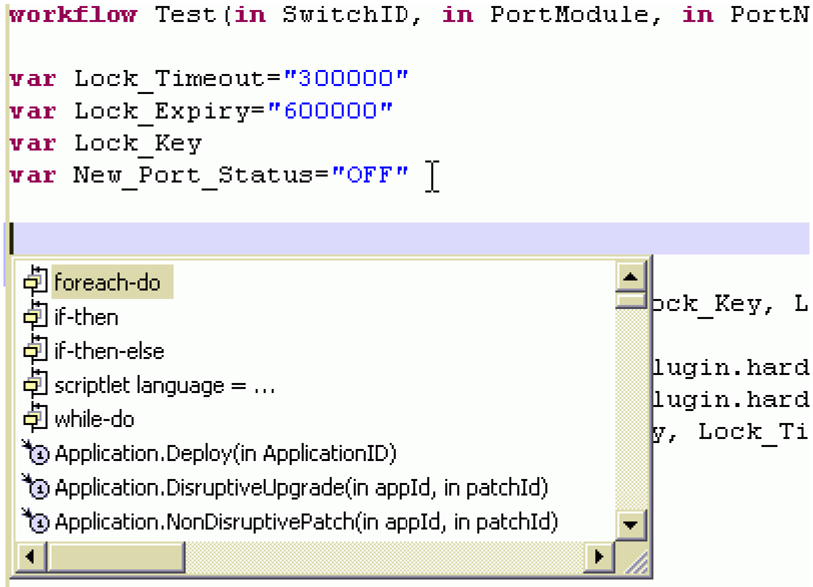
Content assist

The Workflow Composer uses content assist to help you write your workflows. Content assist proposes possible text choices relative to the current position in the document to complete a line in the editor view.

Note: Content assist will only show workflows from the dependent automation package. This is specified when you create the project. You can also modify tc-driver.xml to add more dependencies.

To use content assist in the Workflow Composer:

1. Place your cursor in a valid position on a line of code in the Workflow Composer.
2. Hold down the **Ctrl** key and press the **space bar** on your keyboard. If the Workflow Composer finds valid candidates for this position, a list of possible completions is shown.



```
workflow Test(in SwitchID, in PortModule, in PortN

var Lock_Timeout="300000"
var Lock_Expiry="600000"
var Lock_Key
var New_Port_Status="OFF"
```

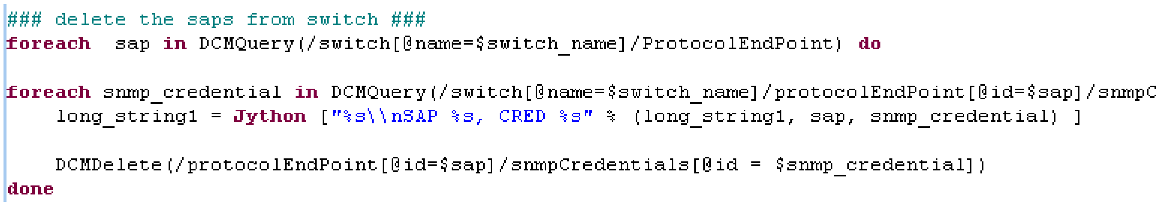
foreach-do
if-then
if-then-else
scriptlet language = ...
while-do
Application.Deploy(in ApplicationID)
Application.DisruptiveUpgrade(in appId, in patchId)
Application.NonDisruptivePatch(in appId, in patchId)
ock_Key, L
lugin.hard
lugin.hard
y, Lock_Ti

Forming data center model query language expressions

The data center model is a representation of all of the physical and logical assets that Tivoli Intelligent Orchestrator manages, such as servers, switches, load balancers, application software, VLANs, and security policies. It keeps track of the data center hardware and associated allocations to applications, as well as changes to configuration.

Using the data center model query language, you can select, insert, update, and delete objects in the data center model. When a workflow that contains a data center model query successfully completes a requested change to the data center, the data center model is updated to reflect the current data center infrastructure.

In the example below, the data center model query language and conditional statements are used to delete the service access points associated with a switch.



```
### delete the saps from switch ###
foreach  sap in DCMQuery(/switch[@name=$switch_name]/ProtocolEndPoint) do

foreach snmp_credential in DCMQuery(/switch[@name=$switch_name]/protocolEndPoint[@id=$sap]/snmpC
    long_string1 = Jython ["%s\\nSAP %s, CRED %s" % (long_string1, sap, snmp_credential) ]

    DCMDelete(/protocolEndPoint[@id=$sap]/snmpCredentials[@id = $snmp_credential])
done
```

For more information on the data center model query language and its syntax , refer to the online help.

Sample workflow: Display the contents of a working directory

The objective of this tutorial is to create a simple workflow that will display the contents of a current working directory using the ls -l command.

Prerequisites: These above requirements should have already been done as part of your Tivoli Intelligent Orchestrator installation

1. Create a server representing the Tivoli Intelligent Orchestrator server. The server name must be the same as the return value from hostname.
2. This server must be configured for an execute-command service access point (SAP).

Step 1: Create a new workflow

1. Click **File** -> **New** -> **Workflow File**
2. Type **DisplayContents** as the name for the new workflow
3. A logical device operation does not have to be selected for this workflow. Click **Finish**.

Step 2: Retrieve the device ID of the Tivoli Intelligent Orchestrator host server

This workflow will run a command on the Tivoli Intelligent Orchestrator host. For that to happen, the Deployment Engine will search for the name of the host on which it is running and will search for a server device ID of that same name. To configure your workflow to do this, add the following line to your workflow:

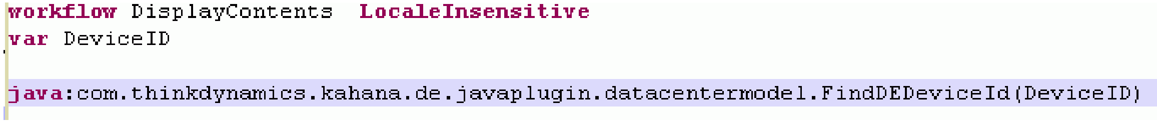
```
java:com.s.kanaha.de.javaplugin.datacentermodel.FindDEDeviceId(DeviceID
)
```

Step 3: Create a local variable for the server's device ID

In the expression above, the Java plug-in is looking for the device ID of the server. To retrieve this ID from the deployment engine, we will need a variable to hold that value. To do this, add the following line to your workflow:

```
var DeviceID
```

Your workflow should now look like this:



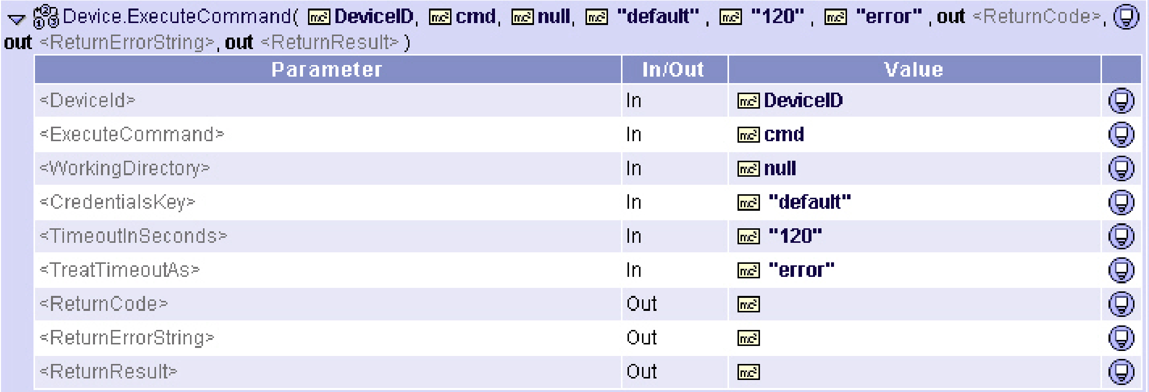
```
workflow DisplayContents  LocaleInsensitive
var DeviceID

java:com.thinkdynamics.kahana.de.javaplugin.datacentermodel.FindDEDeviceId(DeviceID)
```

Step 4: Create a transition to run the ls –l command

The Tivoli Intelligent Orchestrator has a logical device that is designed to run a command that we specify. To add this logical device to your workflow, include the following line in your workflow:

```
Device.ExecuteCommand(DeviceID, cmd, null, "default", "120", "error",
<null>, <null>, <null>)
```



Device.ExecuteCommand(DeviceID, cmd, null, "default" , "120" , "error" , out <ReturnCode>, out <ReturnErrorString>, out <ReturnResult>)

Parameter	In/Out	Value
<DeviceId>	In	DeviceID
<ExecuteCommand>	In	cmd
<WorkingDirectory>	In	null
<CredentialsKey>	In	"default"
<TimeoutInSeconds>	In	"120"
<TreatTimeoutAs>	In	"error"
<ReturnCode>	Out	
<ReturnErrorString>	Out	
<ReturnResult>	Out	

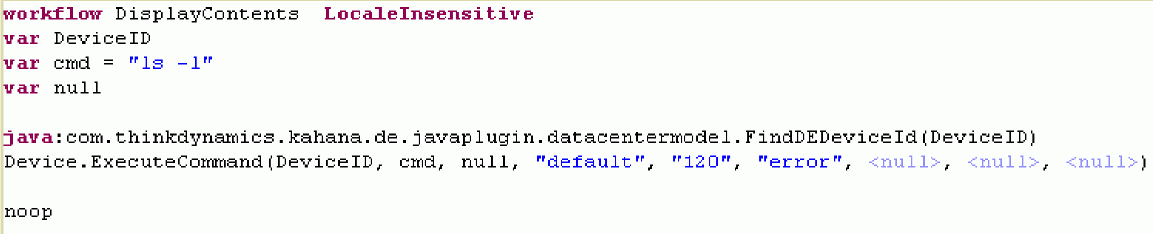
If you look at the properties of the Device.ExecuteCommand, you can see the parameters that need to be defined for the logical device to run. The parameters for our workflow are defined in the image above.

Step 5: Create local variables for the ls -l command

You now need to create local variables for the ls -l command and the working directory. The *ExecuteCommand* and *WorkingDirectory* variables are required variables in the Device.Execute Command transition, so local variables must be created for them. We can also assign the ls –l command directly to our new variable. To do this, add the following lines to your workflow:

```
var null
var cmd = ”ls –l”
cmd = "dir"
```

Your completed workflow should look like this:



```
workflow DisplayContents  LocaleInsensitive
var DeviceID
var cmd = "ls -l"
var null

java:com.thinkdynamics.kahana.de.javaplugin.datacentermodel.FindDEDeviceId(DeviceID)
Device.ExecuteCommand(DeviceID, cmd, null, "default", "120", "error", <null>, <null>, <null>)

noop
```

Step 6: Compile and run your workflow

To compile your workflow, select **Workflow** > **Compile Workflow** from the Eclipse menu.

Sample workflow: Cisco Turn Port ON

Let's examine an existing workflow in the Tivoli Intelligent Orchestrator product for a better understanding of how a workflow is constructed and runs. We will assume, for this example, that a workflow for this operation does not exist and we will need to create it. The Cisco Turn Port ON workflow when run will turn a port associated with a Cisco switch on. To do this, we need to follow five basic steps:

1. Acquire the switch ID associated with the Cisco switch and the port number that we want to turn on.
2. Lock the switch. When you turn a port on, the switch should not be accessible so that it is not interfered with while you are changing its status.
3. Change the port status from OFF (shutdown) to ON (active).
4. Save the current configuration.
5. Unlock the switch.

Note: This tutorial assumes that you have already created a new project for your automation package.

Step 1: Create a new workflow

The first step involves creating a new workflow file and associating that with the logical device operation to turn a switch off. The logical device operation that will do this is: **Switch.TurnPortOFF**. To create a new workflow file:

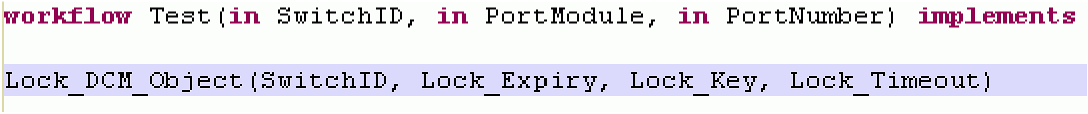
1. Click **File** -> **New** -> **Workflow File**
2. Type a name for the new workflow
3. Expand **Switch**
4. Select **TurnPortOFF**
5. Click **Finish**

Step 2: Lock the switch using an existing workflow

We will have to lock our switch before we can change its state from ON to OFF. To do this, we can use an existing workflow that will lock the data center model object that we want to work with. To insert the Lock_DCM_Object workflow into your new workflow, you can do one of the following:

- If you know the name of the workflow, type it in the workflow composer, including all required variables. You can check the list of workflows in Tivoli Intelligent Orchestrator for this information.
- When you find the workflow that you need, export the workflow file and copy and paste the information into your new workflow. Select Edit > Export within Tivoli Intelligent Orchestrator to export the workflow.

Your workflow should look like this:

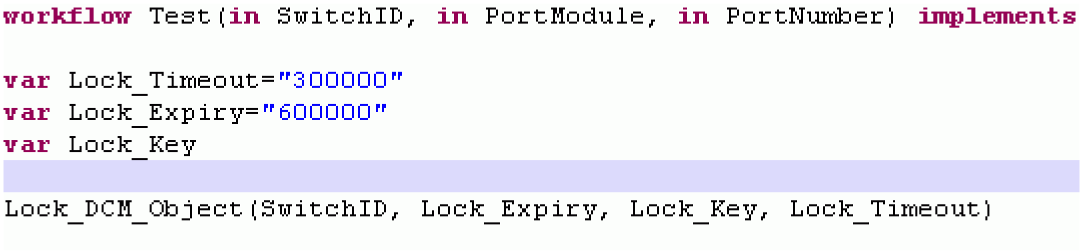


```
workflow Test(in SwitchID, in PortModule, in PortNumber) implements

Lock_DCM_Object(SwitchID, Lock_Expiry, Lock_Key, Lock_Timeout)
```

Step 3: Declare your variables

The workflow that we introduced above has three variables that must be declared in the new workflow. To do this, we will have to declare a variable for the time the port should remain locked and a variable for when that lock state should expire. Assign values to the Lock_Expiry and Lock_Timeout variables. The value should be declared in seconds.



```
workflow Test(in SwitchID, in PortModule, in PortNumber) implements

var Lock_Timeout="300000"
var Lock_Expiry="600000"
var Lock_Key

Lock_DCM_Object(SwitchID, Lock_Expiry, Lock_Key, Lock_Timeout)
```

Step 4: Change the port state from ON to OFF

Creating, configuring, and managing Java plug-ins and logical device operations are required to configure, customize, and successfully run workflows to meet your data center process requirement. Java plug-ins provide the interface for interaction with data center devices. Particular actions performed in the data center can be implemented by the deployment engine using a Java plug-in.

Tivoli Intelligent Orchestrator provides a Java plug-in to change the start of a Cisco switch which can be used with this workflow. To use this Java plug-in, include the following line in the workflow:

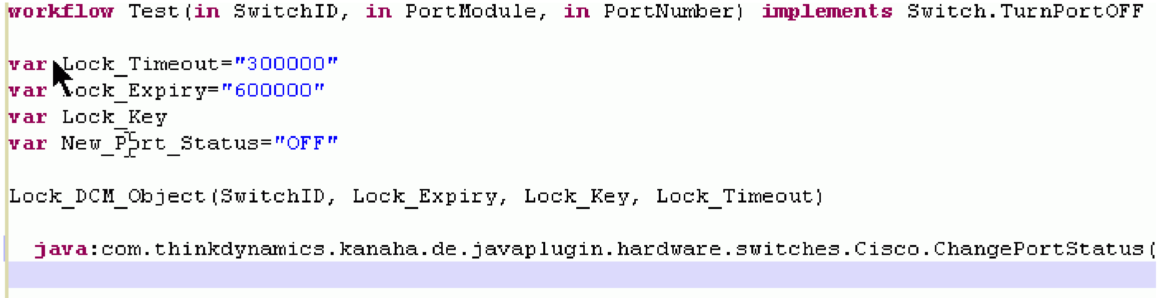
```
java:com.s.kanaha.de.javaplugin.hardware.switches.Cisco.ChangePortStatu
s("write", SwitchID, "1", New_Port_Status, PortNumber)
```

Step 5: Add the New_Port_Status variable

The Java plug-in that was added in Step 3 has introduced a new variable that must be declared in the workflow. Add the **New_Port_Status** variable to your workflow and assign it a value of **OFF**:

var New_Port_Status = "OFF"

Your workflow should now look like this:



```
workflow Test(in SwitchID, in PortModule, in PortNumber) implements Switch.TurnPortOFF

var Lock_Timeout="300000"
var Lock_Expiry="600000"
var Lock_Key
var New_Port_Status="OFF"

Lock_DCM_Object(SwitchID, Lock_Expiry, Lock_Key, Lock_Timeout)

  java:com.thinkdynamics.kanaha.de.javaplugin.hardware.switches.Cisco.ChangePortStatus(
```

Step 6: Save the configuration

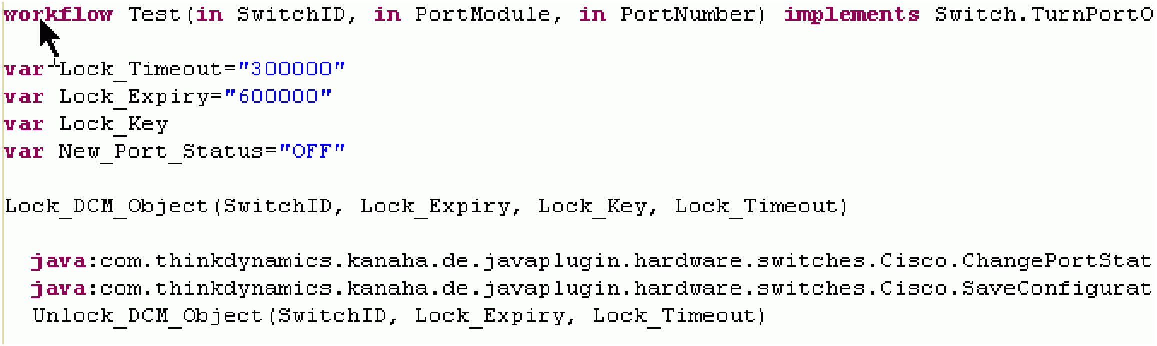
A Java plug-in is available to save your new configuration. Include the following line in your workflow:

```
java:com.s.kanaha.de.javaplugin.hardware.switches.Cisco.SaveConfigurati
on("write", SwitchID)
```

Step 7: Unlock the Cisco switch

To unlock the switch that you had locked in Step 2, we can use an existing workflow that will unlock the data center model object that we want to work with. To insert the Unlock_DCM_Object workflow into your new workflow, follow the steps outlined in Step 2 to retrieve the workflow.

Your completed workflow should look like this:



```
workflow Test(in SwitchID, in PortModule, in PortNumber) implements Switch.TurnPortO

var Lock_Timeout="300000"
var Lock_Expiry="600000"
var Lock_Key
var New_Port_Status="OFF"

Lock_DCM_Object(SwitchID, Lock_Expiry, Lock_Key, Lock_Timeout)

  java:com.thinkdynamics.kanaha.de.javaplugin.hardware.switches.Cisco.ChangePortStat
  java:com.thinkdynamics.kanaha.de.javaplugin.hardware.switches.Cisco.SaveConfigurat
  Unlock_DCM_Object(SwitchID, Lock_Expiry, Lock_Timeout)
```

The automation package

Automation packages are single packages with a **.tcdriver** file extension that contain all of the workflows, database table entries, JAR files, and external scripts that are necessary to operate a physical device, such as a Cisco CSS11000 switch. They represent a logical grouping of entities that together provide a complete solution for a specific device driver.

All automation packages are located in the **$TIO_HOME/drivers** directory on the Tivoli Intelligent Orchestrator or Tivoli Provisioning Manager server. By using automation packages, you can assemble the full suite of logical operations and associate all the behavior of a device-model into one unit.

A default set of automation packages are installed when you install Tivoli Intelligent Orchestrator and Tivoli Provisioning Manager.

When you create a new set of drivers for managing a resource, you should include all the high level logical operations for that device model that apply. Include only those components in the driver that uniquely apply to this solution. There may be other dependencies, but these should be referenced by specifying the dependencies in the tc-driver manifest. For example:

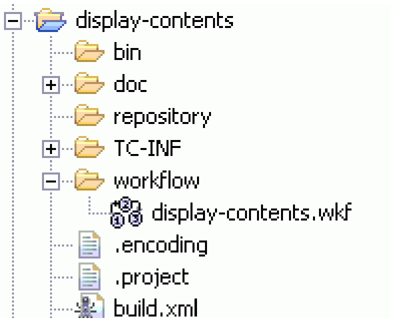
```
<dependencies>
        <dependency name="Core"/>
        <dependency name="Image-Software-Stack"/>
</dependencies>
```

There may not always be a one to one relation in between device drivers and packages. If there are operations that are shared between device drivers consider packaging them together. For example, when packaging the drivers for the load balancer BipIP, the 3.3 and 4.1 versions were packaged together.

Automation package structure

The structure of an automation package includes the following directories:

bin
This directory contains any script files that are run on the deployment engine server. They are not copied to the target.



```
display-contents
    bin
    doc
    repository
    TC-INF
    workflow
        display-contents.wkf
    .encoding
    .project
    build.xml
```

doc
This directory contains the *your_workflow_name*.html file. This file contains information on installing the automation package, as well as information about configuration and troubleshooting.

This directory contains all of the Java plug-ins that are installed and used on that specific physical device.

TC-INF
This directory contains the manifest file for the automation packages.

workflow
This directory contains a collection of workflows that have been developed to operate that specific physical device.

repository
This directory contains scripts that are copied to the target system and run from there.

The tc-driver.xml file

The manifest file for an automation package is an XML file that contains the name and the version number of the automation package, the version number of the automation package template, and describes all of the driver's dependencies on other automation packages. The manifest file must include the following main sections:

<dependencies>	This section lists all of the other drivers the current automation package depends on.
<actions>	This section lists all of the separate classes that are necessary to install separate items like Java drivers, commands, and so on.
<items>	This section lists all the items to be installed on the automation package. Each item identifies a certain operation that will be performed on that automation package.
<device-models>	This section lists all the items to be installed on the automation package. Each item identifies a certain operation that will be performed on that automation package.
<post-install-workflows>	This optional section names a workflow along with its parameters to run after all the items are installed. This workflow could be one that was installed by the current automation package, or one that had been previously installed.
<property>	This optional section defines a macro substitution that can be used for any strings that are referred to in the manifest file. For example, if we have the following entry: <property name="tc.pkg"location="com.s.kanaha.tcdrivermanager.action"/> then, wherever ${tc.pkg} occurs in an attribute string inside tc-driver.xml, a substitution is made.

<software-products>	This section defines any software product entries to install in the data center model database. The syntax is identical to the <software> element in the XML format used by the xmlimport utility, but has been extended to allow for ${xxx}property substitutions within attribute values.
<driver-name>	Name of the driver. This name must exactly match the name of the automation package.
<driver-version>	The version number (optional).
<description>	Describes the purpose of the automation package.
<documentation>	Specifies the name of an html file in the automation package that provides a description of the automation package as well as installation and troubleshooting details.

The order of items in your XML file

The order that the <items> are listed in your XML file is very important and needs to follow a specific pattern. This pattern reflects the order that each item will be installed and uninstalled. Flat files are installed first, followed by Java plug-ins, and workflows. The uninstall of the automation package follows a reverse order: workflows are uninstalled first, followed by Java plug-ins, and all flat files.

Creating the automation package

Step 1: Name your XML file

The name of your XML manifest file must be named: **tc-driver.xml**. A sample tc-driver.xml file is already provided for you. The name of your automation package must be specified between the <driver-name> </driver-name> tags in your tc-driver.xml file. For example: <driver-name>extreme-48i</driver-name>.

Step 2: Create the automation package documentation

An html file has been provided that includes sample details associated with installing an automation package.

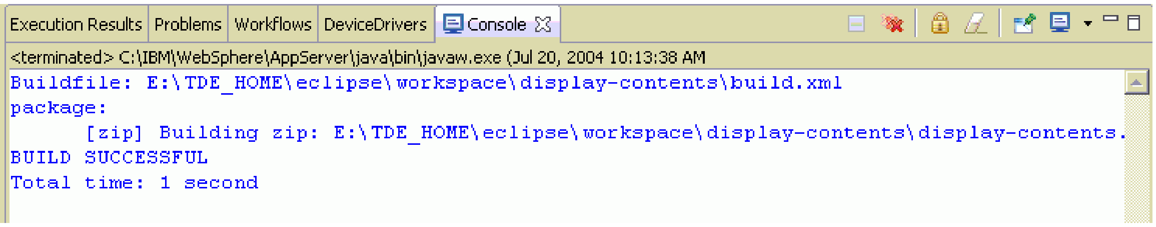
1. Use the sample provided and create the documentation for your automation package.
2. Save the file with a name that matches the name of your automation package: *tcdriver_name*.html. For example, if your automation package is called **extreme-48i.tcdriver**, your html should be named: **extreme-48i.html**.

Package the automation package

Currently, there are a collection of files and folders that make up your automation package. You will have to build this project to make one single .tcdriver.xml file. To do this:

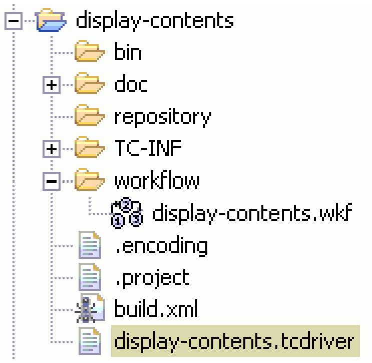
1. In the Navigator view, right-click on the **build.xml** file.

2. Click **Run Ant** .
3. Click **Run**.



```
<terminated> C:\IBM\WebSphere\AppServer\java\bin\javaw.exe (Jul 20, 2004 10:13:38 AM
Buildfile: E:\TDE_HOME\eclipse\workspace\display-contents\build.xml
package:
      [zip] Building zip: E:\TDE_HOME\eclipse\workspace\display-contents\display-contents.
BUILD SUCCESSFUL
Total time: 1 second
```

4. The <***automation package name***>**.tcdriver** file is created and appears in the Navigator view.



Copy and install the driver

1. Export the <***automation package name***>**.tcdriver** file to the **%TIO_HOME%/drivers** directory.
2. Change the directory to: **%TIO_HOME%/tools**
3. Run `tc-driver-manager.cmd installDriver <automation package name>`. On non-Windows systems, run the `tc-driver-manager.sh` command.