

# HP Service Activator

## Workflows and the Workflow Manager

Edition: V70-1A

for Microsoft Windows® Server 2012 R2, HP-UX 11i v3,  
and Red Hat Enterprise Linux 6.6 operating systems



**Manufacturing Part Number: None**

**January 18, 2015**

© Copyright 2001-2015 Hewlett-Packard Development Company, L.P.

---

## Legal Notices

### **Warranty.**

*Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.*

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office

### **Restricted Rights Legend.**

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company  
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

### **Copyright Notices.**

©Copyright 2001-2015 Hewlett-Packard Development Company, L.P.

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Printed in the US.

**Trademark Notices.**

Java™ is a registered trademark of Oracle and/or its affiliates.

Linux is a U.S. registered trademark of Linus Torvalds.

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

Red Hat® Enterprise Linux® is a registered trademark of Red Hat, Inc.

JBoss® is a registered trademark of Red Hat, Inc. in the United States and other countries.

EnterpriseDB® is a registered trademark of EnterpriseDB.

Postgres Plus® Advanced Server is a registered trademark of EnterpriseDB.

Oracle® is a registered trademark of Oracle and/or its affiliates.

UNIX® is a registered trademark of the Open Group.

Windows® and MS Windows® are U.S. registered trademarks of Microsoft Corporation.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

Document id: p158-pd001510



**1. Understanding Workflows and Workflow Manager**

What Is A Workflow? .....	20
Workflow Nodes .....	20
Understanding the Workflow Manager .....	22
Workflow Manager Modules .....	22
Programming Analogy .....	24

**2. Creating and Deploying Workflows**

Understanding Workflows .....	26
General Structure of Workflow .....	26
Startup Attributes .....	27
Workflow Persistence Attribute .....	27
Audit and Statistics Attributes .....	28
Setting Roles .....	29
Workflow Nodes .....	30
Handlers .....	33
Conventions for Node and Handler Parameters .....	34
Case-Packet Variables .....	35
Initial Case-Packet Values .....	35
Workflow Contract .....	36
Default Case-Packet Variables .....	36
References to Complex Data Types in Workflow Node Parameters .....	43
Queues .....	46
Advanced Workflow Techniques .....	47
Spawning Child Workflows .....	47
Using Timeouts .....	51
Using Prioritization in Workflows .....	53
Uploading Data from a Task Activation .....	56
Deploying Workflows .....	57
Clustering Considerations .....	58

**3. Using the Workflow Designer**

Navigating the Workflow Designer .....	60
Understanding Workflow Designer Features .....	61
Using the Main Menu .....	63
Using the Main Utilities Toolbox .....	66
Using the Visual Properties Toolbox .....	67
Using the Context Sensitive Menu .....	69
Using the Workflow Views .....	71
Copying and Pasting Workflow Nodes .....	74
Deleting Nodes .....	76
Using the Node Tree .....	78
Using the Overview Pane .....	80
Using the Node Properties View .....	81
Lock / Unlock Function .....	82
Using the Edit Node Properties Dialog .....	83

---

# Contents

Using the Action Parameters Tab .....	84
Command Line Options .....	86
Using Keyboard Shortcuts .....	87

## 4. Workflow Node and Handler Library

Process Nodes, Rule Nodes, and Switch Nodes .....	90
Default Workflow Node Persistence Setting .....	90
Activate .....	96
Add .....	100
AppendToResourceList .....	101
AppendToTaskList .....	102
AskFor .....	104
Assign .....	109
Audit .....	110
ChangeRoles .....	113
ComposeMessage .....	114
ConcatenateTaskLists .....	117
ConfirmResourceReservation .....	118
CreateBean .....	119
CreateInventory .....	121
CreateTaskList .....	123
CreateUCMDBCIAndRelations .....	125
DateConverter .....	129
Decrypt .....	133
DeleteCache .....	134
DeleteInventory .....	135
DeleteScheduledJob .....	136
DeleteServiceInstance .....	137
DeleteUCMDBCIAndRelations .....	138
DoNothing .....	140
Encrypt .....	141
ExecuteMacro .....	142
Equal .....	144
ExecSQLQuery .....	145
ExecSQLStatement .....	147
ExecuteExternal .....	148
ForEach .....	150
GenericUIDialog .....	152
GetBaseFileName .....	159
GetBeansNNMNode .....	160
GetBusinessHoursAfterDuration .....	165
GetCalendarTimezone .....	168
GetNextIncludedTime .....	169
GetTimeRangesOfBusinessDay .....	171
GetOperatingSystem .....	173
GreaterThan .....	174

GreaterThanOrEqualTo	175
HTTPGet	176
HTTPRequest	179
InsertIntoTasklist	181
InvokeInventoryMethod	183
InvokeMethod	184
IsModuleConfigured	187
IsTimeIncluded	188
IsTrue	190
Java	191
JavaRule	196
JavaSwitch	198
KillJob	201
LessThan	202
LessThanOrEqualTo	203
Log	204
MapData	206
MatchDBQuery	208
MatchDBStore	210
MethodRule	212
ModifyScheduledJob	213
MoveFile	215
MultiAssign	216
Multiply	217
MutexGetInfo	218
MutexLock	219
MutexSetInfo	221
MutexUnlock	222
NAAddConfigurationPolicy	223
NAAddDevice	225
NAAddDeviceGroup	226
NAAddDeviceToGroup	227
NAAddRuleToPolicy	228
NABuildConditionList	229
NABuildRuleList	230
NADeleteDeviceGroup	231
NADeletePolicy	232
NAGetSnapshot	233
NAListConfigId	234
NAListDevice	235
NAModifyConditionsOnRule	236
NARemoveDeviceFromGroup	237
NARemoveRuleFromPolicy	238
NARunAdvancedScript	239
NARunCommandScript	240
NARunScript	241

---

# Contents

NAShowConfig . . . . .	242
NAShowDiagnostic . . . . .	243
NAShowTask . . . . .	244
Not . . . . .	245
PAYG . . . . .	246
PatternMatch . . . . .	247
PPU . . . . .	248
PutMessage . . . . .	249
QueryInventory . . . . .	253
QueryScheduledJob . . . . .	256
QueryServiceInstance . . . . .	259
QueryServiceInstanceAll . . . . .	261
QueryUCMDBCsAndRelations . . . . .	262
RandomInteger . . . . .	267
ReadData . . . . .	268
ReadDataFromDatabase . . . . .	271
ReadFile . . . . .	274
RediscoverHostsNNMNode . . . . .	275
ReleaseResource . . . . .	276
RemoveData . . . . .	277
RemoveFile . . . . .	279
Replace . . . . .	280
ReserveResource . . . . .	281
RetrieveSequence . . . . .	284
ScheduleCurrentJob . . . . .	285
ScheduleJob . . . . .	286
SendAlarm . . . . .	289
SendMessage . . . . .	291
SendSNMPTrap . . . . .	293
Sleep . . . . .	294
StartJob . . . . .	295
StartJobAndWait . . . . .	297
Switch . . . . .	299
Sync . . . . .	300
ThrowError . . . . .	302
ThrowException . . . . .	303
ThrowRuntimeException . . . . .	304
TransformXML . . . . .	305
UpdateBean . . . . .	311
UpdateCustomAttributesNNMNode . . . . .	313
UpdateInProgress . . . . .	314
UpdateInventory . . . . .	315
UpdateServiceInstance . . . . .	318
UpdateUCMDBCsAndRelations . . . . .	319
VariableMapper . . . . .	323
WasPreviousNodeOK . . . . .	325



WriteCasePacket . . . . .	326
WriteDataToDatabase . . . . .	327
XMLMapper . . . . .	329
XMLParser . . . . .	334
Handlers . . . . .	339
ComposeMessageHandler . . . . .	340
MultiAssignHandler . . . . .	342
DoNothingHandler . . . . .	343
PutMessageHandler . . . . .	344
ReleaseResourceHandler . . . . .	345
SendMessageHandler . . . . .	346
SyncHandler . . . . .	347
VariableMapperHandler . . . . .	349

## **5. Configuring the Workflow Manager**

Setting the Workflow Manager Parameters . . . . .	352
Understanding Workflow Manager Modules . . . . .	356
Required and Typical Workflow Manager Modules . . . . .	356

## **6. Workflow Manager Module Library**

Using the Workflow Manager Module Library . . . . .	362
ActivationModule . . . . .	363
AuditModule . . . . .	365
BusinessCalendarModule . . . . .	367
CacheModule . . . . .	368
CasePacketDistModule . . . . .	369
CheckTimeModule . . . . .	371
ConflictModule . . . . .	373
DatabaseAdvancedAuthModule . . . . .	374
DBTransactionModule . . . . .	377
HPUXAdvancedAuthModule . . . . .	378
HTTPRenderer . . . . .	380
HTTPSenderModule . . . . .	381
JMSListenerModule . . . . .	383
JMSSenderModule . . . . .	387
JNDIDatabaseModule . . . . .	389
KeepAliveModule . . . . .	390
LDAPAuthModule . . . . .	394
LinuxAdvancedAuthModule . . . . .	399
LoadFactorDistModule . . . . .	401
LogSearchModule . . . . .	403
MailHook . . . . .	405
Monitor . . . . .	407
NARequestModule . . . . .	409
NNMRequestModule . . . . .	412
OVOMessageModule . . . . .	415

---

# Contents

QueueDistModule . . . . .	416
QueueDBPersistenceModule . . . . .	418
QueueModule . . . . .	419
RoundRobinDistModule . . . . .	421
SchedulerModule . . . . .	422
SelfMonitoringModule . . . . .	424
SNMPSenderModule . . . . .	426
SocketListenerModule . . . . .	428
SocketSenderModule . . . . .	432
SolutionXMLLogModule . . . . .	434
SyncModule . . . . .	436
UCMDBRequest Module . . . . .	441
UsageMonitoringModule . . . . .	443
WindowsAdvancedAuthModule . . . . .	444
WorkManagerModule . . . . .	447
XMLLogModule . . . . .	448

## 7. Writing Custom Workflow Nodes

Understanding Workflow Nodes . . . . .	452
Accessing Workflow Manager Capabilities: WFContext & WFManager . . . . .	453
Example Source Code for Nodes . . . . .	453
Writing Custom Process Nodes . . . . .	454
Writing Custom Rule Nodes . . . . .	458
Writing Custom Switch Nodes . . . . .	460
Writing Error and End Handlers . . . . .	461
Deploying Workflow Nodes and Handlers . . . . .	462
Using Custom Nodes and Handlers in Designer . . . . .	462

## 8. Writing New Workflow Modules

Writing New Workflow Manager Modules . . . . .	466
Example Source Code for Modules . . . . .	466
Implementation of Modules . . . . .	467
Master-Slave . . . . .	471
Writing New Authenticator Module . . . . .	473
Writing New Queue Hook . . . . .	476
Writing New Sender Module . . . . .	477
Writing New Message Module . . . . .	478
Deploying Workflow Manager Modules . . . . .	479

## 9. Writing Workflow Manager Clients

Writing Workflow Manager External Interface Clients . . . . .	482
Creating a Workflow Manager Client . . . . .	483
Examples . . . . .	484

## A. Configuring Service Activator to Use Secure Socket Layer (SSL) Protocol

Using SSL with Service Activator: An Overview . . . . .	488
---	-----

Preparing to Use SSL . . . . .	488
Getting Organized . . . . .	488
Configuring Service Activator to Use SSL . . . . .	488
Understanding the Required Software . . . . .	489
Configuring JSSE . . . . .	489
Preparing to Load the Certificate Keystore . . . . .	489
Managing Keys and Certificates . . . . .	490
Configuring SSL for HTTPS (Operator UI) . . . . .	492
Step 1: Loading the Server Keystore (Operator UI) . . . . .	492
Step 2: Modifying the JBoss Configuration Files . . . . .	492
Step 3: Starting Service Activator . . . . .	493
Configuring SSL for Secure Message Transmission (Workflow Manager) . . . . .	494
Step 1: Loading the Server Keystore (Workflow Manager) . . . . .	494
Step 2: Modifying the Workflow Manager Configuration File . . . . .	494
Step 3: Restarting the HP Service Activator . . . . .	495
Troubleshooting . . . . .	496
Finding Additional Information . . . . .	497
<b>B. mwfmtool</b>	
mwfmtool . . . . .	500

## **C. Creating Additional Data Source**



---

## In This Guide

This guide describes the HP Service Activator Workflow Manager and the workflows required to use Service Activator.

### **Audience**

The audience for this guide is the Solutions Integrator (SI). The SI has a combination of some or all of the following capabilities:

- Understands and has a solid working knowledge of:
  - UNIX® commands
  - Windows® system administration
- Understands networking concepts and language
- Is able to program in Java™ and XML
- Understands security issues
- Understands the customer's problem domain



## Conventions

The following typographical conventions are used in this guide.

Font	What the Font Represents	Example
<i>Italic</i>	Book or manual titles, and manpage names	Refer to the <i>HP Service Activator — Workflows and the Micro-Workflow Manager</i> and the <i>Javadocs</i> manpage for more information.
	Provides emphasis	You <i>must</i> follow these steps.
	Specifies a variable that you must supply when entering a command	Run the command: InventoryBuilder <sourceFiles>
	Parameters to a method	The <i>assigned_criteria</i> parameter returns an ACSE response.
<b>Bold</b>	New terms	The <b>distinguishing attribute</b> of this class...
Computer	Text and items on the computer screen	The system replies: Press Enter
	Command names	Use the InventoryBuilder command ...
	Method names	The get_all_replies() method does the following...
	File and directory names	Edit the file \$ACTIVATOR_ETC/config/mwfm.xml
	Process names	Check to see if mwfm is running.
	Window/dialog box names	In the Test and Track dialog...
	XML tag references	Use the <DBTable> tag to...
<b>Computer Bold</b>	Text that you must type	At the prompt, type: <b>ls -l</b>
<b>Keycap</b>	Keyboard keys	Press <b>Return</b> .
[Button]	Buttons on the user interface	Click [Delete]. Click the [Apply] button.

<b>Font</b>	<b>What the Font Represents</b>	<b>Example</b>
Menu Items	A menu name followed by a colon (:) means that you select the menu, then the item. When the item is followed by an arrow (->), a cascading menu follows.	Select Locate:Objects->by Comment



## Install Location Descriptors

The following names are used throughout this guide to define install locations.

Descriptor	What the Descriptor Represents
<p><i>\$ACTIVATOR</i> <i>\$ACTIVATOR_OPT</i></p>	<p>The base install location of Service Activator. The UNIX location is /opt/OV/ServiceActivator The Windows location is &lt;install drive&gt;:\HP\OpenView\ServiceActivator\</p>
<p><i>\$ACTIVATOR_ETC</i></p>	<p>The install location of specific Service Activator files. The UNIX location is /etc/opt/OV/ServiceActivator The Windows location is &lt;install drive&gt;:\HP\OpenView\ServiceActivator\etc\</p>
<p><i>\$ACTIVATOR_VAR</i></p>	<p>The install location of specific Service Activator files. The UNIX location is /var/opt/OV/ServiceActivator The Windows location is &lt;install drive&gt;:\HP\OpenView\ServiceActivator\var\</p>
<p><i>\$ACTIVATOR_BIN</i></p>	<p>The install location of specific Service Activator files. The UNIX location is /opt/OV/ServiceActivator/bin The Windows location is &lt;install drive&gt;:\HP\OpenView\ServiceActivator\bin\</p>
<p><i>\$JBOSS_HOME</i></p>	<p>The install location for JBoss. The UNIX location is /opt/HP/jboss The Windows location is &lt;install drive&gt;:\HP\jboss</p>
<p><i>\$JBOSS_DEPLOY</i></p>	<p>The install location of the Service Activator J2EE components. The UNIX location is /opt/HP/jboss/standalone/deployments The Windows location is &lt;install drive&gt;:\HP\jboss\standalone\deployments</p>
<p><i>\$JBOSS_EAR_LIB</i></p>	<p>The location for libraries (Java *.jar files) to be executed by the HPSA engine (workflow manager and resource manager). The UNIX location is /opt/HP/jboss/standalone/deployments/hpsa.ear/lib The Windows location is &lt;install drive&gt;:\HP\jboss\standalone\deployments\hpsa.ear\lib</p>
<p><i>\$ACTIVATOR_DB_USER</i></p>	<p>The database user name you define. Suggestion: ovactivator</p>
<p><i>\$ACTIVATOR_SSH_USER</i></p>	<p>The Secure Shell user name you define. Suggestion: ovactusr</p>



---

# **1      Understanding Workflows and Workflow Manager**

## What Is A Workflow?

A workflow generally represents some business process or activation process that is being automated. In a workflow, the process is broken down into discrete steps called nodes. The workflow executes these nodes in the proper sequence and allows conditional branching within the process.

Business processing steps that you can automate using workflow nodes include activities such as:

- Extracting data from an incoming XML message.
- Calculating derived parameters.
- Requesting and updating data from external repositories such as an inventory database.
- Sending messages to external processes.
- Waiting for input from a human operator or external process.
- Activation—performing hardware or software configuration according to gathered parameters.

A workflow always has a collection of variables called the **case-packet**. The case-packet maintains the state of the workflow. The nodes in the workflow examine and may update the values in the case-packet.

Since a workflow sounds very similar to a computer program, you may wonder why you might choose to write a workflow rather than encoding your business process in another programming language or scripting language. You should keep in mind that workflows generally represent potentially long running business processes. The business process must continue across stops and restarts of a machine. To this end, the Workflow Manager is able to make the state of a running job persistent and to resume the job at any point. Additionally, these jobs may require external interaction (with a user, for example) that may not happen in a predetermined amount of time. Workflows are capable of posting a request for input and then effectively removing themselves from active processing until the external process (or user) sends the needed input.

## Workflow Nodes

A workflow consists of a sequence of workflow nodes. Workflow nodes, which are implemented by Java classes, provide the operational logic that allows Service Activator to interact with many sources of information and perform various tasks.

There are three categories of nodes:

- **Process nodes**, which perform some tasks and typically change the state of the workflow case-packet.
- **Rule nodes**, which perform a simple test that causes a branch in the workflow.
- **Switch nodes**, which select one of multiple branches based on a simple test

Most business process and activation tasks can be accomplished using the process, rule, and switch nodes from the library provided with Service Activator. However, it is possible to implement your own nodes to perform complex business logic or to interact with a different information source.

For more information on workflow nodes, see Chapter 4, “Workflow Node and Handler Library,” on page 89. To create your own nodes, see Chapter 7, “Writing Custom Workflow Nodes,” on page 451.

## Understanding the Workflow Manager

The Workflow Manager, is a program that provides the ability to run multiple— even thousands of—simultaneous workflows. As mentioned earlier, the Workflow Manager maintains the state of all running workflows and takes care of the processing necessary to enable the workflows to be restarted should the Workflow Manager be shutdown (gracefully or catastrophically).

The open architecture of the Workflow Manager allows extension of this engine in two ways:

- To allow communication with external systems using mechanisms other than those provided with Service Activator. This is done via configurable Workflow Manager modules.
- To specify custom functionality for applying business logic to steps in the activation process. This is done by writing new workflow nodes.

You can configure the Workflow Manager using specific parameters or by adding and configuring Workflow Manager modules. All of this configuration takes place in the file `§ACTIVATOR_ETC/config/mwfm.xml`. When the product is first installed, the Workflow Manager is given an initial configuration that will be appropriate during development and in some customer installations. For most customer installations, additional configuration will be necessary. For details see Chapter 5, “Configuring the Workflow Manager,” on page 351.

### Workflow Manager Modules

Workflow Manager modules are Java classes that provide communication between external processes and systems and the Workflow Manager. These modules provide functionality such as:

- Authenticating users.
- Receiving and sending XML documents (via sockets or on a bus, for example).
- Providing access to a database.
- Monitoring the Workflow Manager and making those statistics available externally.
- Sending messages to HP OpenView Operations (OVO).
- Sending e-mail.
- Logging actions that the Workflow Manager performs.
- Gaining access to the activation engine, which is a separate piece in the Service Activator architecture.

You can also implement new Workflow Manager modules to extend the reach of Service Activator. For example, you might want to provide an interface between the Workflow Manager and a new communication bus.

Modules must be configured before they can be used by the Workflow Manager. Some modules must also be configured to work properly in the environment (the `DatabaseModule`, for example, must identify the proper database with which to communicate).

See “Understanding Workflow Manager Modules” on page 356 for general information about modules. For more information on the Workflow Manager modules provided with Service Activator, see “Using the Workflow Manager Module Library” on page 362. To create your own modules, see Chapter 8, “Writing New Workflow Modules,” on page 465.

## Programming Analogy

A programming analogy is helpful for understanding the Workflow Manager and workflows:

- The Workflow Manager is analogous to an operating system (OS). It runs as a JBoss service inside JBoss.
- A workflow is analogous to an interpreted, executable program. A workflow can be depicted like a flowchart. A workflow exists on disk as an XML file when used by the Workflow Designer, but in the database when the workflow is deployed
- A running workflow (a job) is analogous to a process within the OS.
- Nodes are analogous to a library of procedures that can be called from the workflow. Nodes can be depicted as steps in the flowchart. A node can be configured to behave in a certain way within each workflow and can be used multiple times within the same workflow (and differently each time).
- A case-packet is analogous to a set of global variables. All of the variables in the case-packet are accessible (readable and writable) by all of the nodes in the workflow.
- Modules are analogous to device drivers in that a driver usually adheres to an abstract interface but has a concrete implementation that knows how to communicate with a specific outside entity.



---

## **2** **Creating and Deploying Workflows**

This chapter describes the structure of a workflow definition and explains how workflows are deployed in a Service Activator installation.

## Understanding Workflows

Everything about a workflow is defined within the XML file. Because a workflow is fully defined in an XML file, you can create a workflow using your favorite text editor. However, there is a graphical tool that greatly simplifies the process of writing workflows. The Workflow Designer tool is described in Chapter 3, “Using the Workflow Designer,” on page 59.

The rest of this chapter describes the XML structure of a workflow.

### General Structure of Workflow

A workflow definition includes the following parts:

- The name of the workflow.
- The name of the solution the workflow belongs to (optional).
- A description of the workflow (optional).
- Role settings for who may perform various actions with respect to this workflow (optional).
- Which node in the workflow to start with.
- The collection of nodes in the workflow.
- Error handlers indicating what to do in the case of an exception being raised during the processing of the workflow (optional).
- End handlers indicating what to do at the end of the workflow (optional).
- The collection of case packet variables.
- The initial case packet values (optional).
- Graphical layout details (optional).
- The workflow contract

To take full advantage of solution separation then the workflow should include the solution name. For more details see the document “Solution Separation and the Deployment Manager”.

See the `workflow.dtd` file (in `$ACTIVATOR_ETC/workflows/`) for details about the exact ordering of all possible tags. A general XML workflow structure looks similar to this:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE Workflow SYSTEM "workflow.dtd">
<Workflow>
  <Name>ExampleWorkflow</Name>
  <Solution>Example</Solution>
  <Description>Example showing the structure of a workflow</Description>
  <Default-Role>operator</Default-Role>
  <Start-Node>Initial node</Start-Node>
  <Nodes>
    . . .
  </Nodes>
  <Case-Packet>
    . . .

```

```
</Case-Packet>  
<Initial-Case-Packet>  
  * * *  
</Initial-Case-Packet>  
</Workflow>
```

---

**NOTE**

If you create a workflow in a text editor, your workflow will not include any graphical layout information. Once the workflow has been edited in the Workflow Designer, the workflow will include graphical layout details at the end of the workflow. These can usually be ignored when editing manually. However, if you delete nodes or change their names, the layout details will become inconsistent with the rest of the workflow, and you may need to completely delete the layout details before the workflow can again be edited with the Workflow Designer.

---

## Startup Attributes

There are two optional startup attributes that you can set as part of the `<Workflow>` tag. They are:

- `Init-On-Startup`. When set to “true,” it starts the workflow automatically (without user intervention) when the Workflow Manager starts. The default is “false”.
- `Unique`. When set to “true,” it restricts the use of the workflow to one instance at a time. The default is “false”.

To set both of these attributes to “true,” use the following structure in the `<Workflow>` tag:

```
<Workflow Init-On-Startup="true" Unique="true">
```

---

**NOTE**

Although you can set these attributes individually, it is best to use the `Unique` attribute (set to “true”) whenever you set the `Init-On-Startup` to “true.” Otherwise, if the Workflow Manager shuts down unexpectedly, another workflow of the same type is launched when the system restarts. Set the `Unique` option to “true” to prevent more than one instance of a workflow when the `Init-On-Startup` attribute is set to “true.”

---

## Workflow Persistence Attribute

The `disablePersistence` attribute indicates if persistence of the workflow should be done. If `disablePersistence` is set to `true` the workflow will not be stored during its execution, which means that if Service Activator is stopped while the workflow is running it will not be started again when Service Activator is restarted. This will also be the case even if the transaction module is configured. The default value is “false”.

Here is an example of how the attribute is used:

```
<Workflow disablePersistence="false">
```

## Audit and Statistics Attributes

There are three optional attributes which control collecting additional data for a workflow. They can be set as part of the `<Workflow>` tag. Below are the three attributes:

- **auditEnabled.** If this attribute is set to “false”, audit records are not collected for the workflow. The default value is “true”.
- **autoAuditEnabled.** Set this attribute to “false”, if you do not want to collect automatically generated audit.
- **statEnabled.** If this attribute is set to “false”, statistical records are not collected for the workflow. The default value is “true”.

Here is an example of how the three attributes are used:

```
<Workflow auditEnabled="false" autoAuditEnabled="true" statEnabled="true">
```

## Setting Roles

Roles specify who can perform a given operation or interact with a workflow.

The following roles can be specified for the entire workflow:

- Default role (`<Default-Role>...</Default-Role>`). Indicates who is allowed to perform the `Start/Trace/Kill` functions if not otherwise specified. It also indicates who can interact with the workflow or see messages from the workflow. See the `AskFor` and `PutMessage` nodes for more description of how roles affect these nodes.
- Start role (`<Start-Role>...</Start-Role>`). Indicates who is allowed to start this workflow. Anyone in the given role can start the workflow. If this is not specified, this takes the value of the `<Default-Role>`. This only applies when someone tries to start the workflow from the command line or from the operator UI.
- Trace role (`<Trace-Role>...</Trace-Role>`). Indicates who is allowed to view the current state of the workflow. If this is not specified, it takes the value of the `<Default-Role>`.
- Kill role (`<Kill-Role>...</Kill-Role>`). Indicates who is allowed to kill this workflow. If this is not specified, it takes the value of the `<Trace-Role>`.

You can also specify a role in an individual node to indicate which users are allowed to interact with or receive a message from that node. This node setting overrides the `Default-Role` that is set in the workflow heading. Generally this is only appropriate for those nodes that perform an interactive task such as `AskFor` or `PutMessage`. The syntax for this node setting is described in “Process Nodes” on page 31.

Roles are statically set in the workflow, but can be changed dynamically on individual jobs either from the `ChangeRoles` node or from the UI as administrator.

---

### NOTE

Service Activator also provides the ability to do advanced role mapping, making it possible to write a workflow in a generic way using logical or virtual role names. These generic names can then be mapped to the real roles or groups that are meaningful in the customer’s environment. See the following sections for more information on roles:

- “Authentication” on page 357.
- “Roles, Privileges and Authentication” in *HP Service Activator System Integrator’s Overview*.

---

### NOTE

To stop a job from the Operator UI, the user must be assigned to both the `Trace-Role` and the `Kill-Role`.

## Workflow Nodes

Workflow nodes carry out the work of a workflow. You should notice that a node always has a `<Name>` and an `<Action>`. The `<Name>`, which is unique within the workflow, is simply the handle to this node within the workflow. The `<Action>` is what defines the behavior of the step in the workflow.

A workflow node has one optional attribute `disablePersistence`. If this attribute is set to `"true"` no persistence will be done after the processing of the node. The default is `"false"`. If persistence is done a bobble will be displayed in the upper left corner of the icon of the node in the Workflow Designer.

Here is an example of how the the attribute is used:

```
<Rule-Node disablePersistence="true">
```

All nodes delivered with Service Activator are configured with a default behaviour. The general behavior of Process Nodes is to persist where the Rule and Switch Nodes do not persist. A table about which nodes persist can be found in a table before the description of each node.

The general behavior of the node is indicated by the `<Action><Class-Name>` tag. This indicates a Java class that implements the behavior of the node. Although each node has a general behavior, it must usually be configured for the specific behavior that should occur at a given step in the workflow. This specific behavior is specified in the parameters of the action. Each parameter has a name and a value. For example, a parameter with the name `"task"` is used by the `Activate` node. The value of the `task` parameter is the name of the atomic or compound task to be activated. Each node class supports a different set of parameters. These parameters are described in detail in Chapter 4, "Workflow Node and Handler Library," on page 89.

There are three categories of nodes:

- **Process nodes** perform activities such as querying the inventory, asking a human operator for information, invoking the activation of a compound task, and so on. They are represented as rectangles in the Workflow Designer.
- **Rule nodes** are the decision nodes that branch the workflow depending on given conditions. For example, if a given condition is true, the workflow branches one direction; if false, it branches a different direction. Rule nodes are depicted as diamonds in the Workflow Designer.
- **Switch nodes** are very similar to Rule nodes. A switch node can have multiple branches but requires at least a default branch. The switch node evaluates its expression and executes the appropriate case where case is the same as a branch. For example, if a given expression evaluates to 5, the workflow select the case with the value 5; if a case cannot be found the default branch is selected. Switch nodes are depicted as parallelograms in the Workflow Designer.

## Process Nodes

The general structure of a Process Node definition in a workflow looks like this:

```
<Process-Node>
  <Name>Create a new directory</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.engine.component.builtin.Activate
    </Class-Name>
    <Param name="task" value="UXOS_addDir"/>
    <Param name="param0" value="machine"/>
    <Param name="param1" value="dirname"/>
    <Param name="param2" value="login"/>
    <Param name="param3" value="constant:users"/>
    <Param name="param4" value="constant:775"/>
    <Param name="param5" value="tarfile"/>
  </Action>
</Process-Node>
```

Process nodes optionally have a `<Role>`, `<State>`, and a `<Next-Node>` tag. The role indicates which user or users can interact with the node or receive messages from the node. The role of a node overrides the `Default-Role` setting in the workflow. Setting the role of a node is only meaningful if the node performs some user interaction or sends a message. The state of a node is a value that can be used to group a set of workflow nodes into coarse-grained states. The `<Next-Node>` tag indicates which node in the workflow will be processed after this one. If no `<Next-Node>` tag is present, then this node represents an end node of the workflow.

## Inactive Process Nodes

Without changing the structure of a workflow you can tell the Workflow Manager to skip over individual process nodes in the workflow. This may be helpful during development when testing your workflows. It may also be helpful after development is completed and you want to eliminate some processing from the workflow that is not necessary when the workflow is running in a production environment. The syntax for this is:

```
<Process-Node inactive="true">
```

This can be accomplished in the Workflow Designer, by setting the Inactive Node flag in the context sensitive pop-up menu associated with a node.

## Test Process Nodes

Very similar to Inactive Process Nodes. The difference is that the process node is skipped if the parameter `Test-Mode` is set to false in the `mwfm.xml` configuration file and executed if set to true. The syntax for this is:

```
<Process-Node test="true">
```

This can be accomplished in the Workflow Designer, by setting the Test Node flag in the context sensitive pop-up menu associated with a node.

## Rule Nodes

The general structure of a Rule Node definition in a workflow looks like this:

```
<Rule-Node disablePersistence="true">
  <Name>CheckSuccess</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Equal
    </Class-Name>
    <Param name="op0" value="RET_VALUE"/>
    <Param name="op1" value="constant:0"/>
  </Action>
  <True-Next-Node>CheckSuccess</True-Next-Node>
  <False-Next-Node>CheckSuccess</False-Next-Node>
</Rule-Node>
```

The structure of a Rule Node is similar to a Process Node. The distinction is that a Rule Node must have a `<True-Next-Node>` and a `<False-Next-Node>` to indicate which node should be processed next in the flow depending on the outcome of the test.

Rule nodes may not have a `<Role>` tag.

## Switch Nodes

The general structure of a SwitchNode definition in a workflow looks like this:

```
<Rule-Node disablePersistence="true">
  <Name>CheckSwitch</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.SwitchCase
    </Class-Name>
    <Param name="key" value="value"/>
    <Param name="case0" value="constant:red"/>
    <Param name="case1" value="constant:green"/>
  </Action>
  <Switch name="case0">PutMessage1</Switch>
  <Switch name="case1">PutMessage2</Switch>
  <Default>PutMessage3</Default>
</Rule-Node>
```

The structure of a SwitchNode is similar to a Process Node. The distinction is that a Switch Node must have a `<Default>` tag and a optional number of `<Switch>` tag to indicate which node should be processed next in the flow depending on the outcome of the test.

Switch nodes may not have a `<Role>` tag.



## Handlers

These workflow elements run at the completion of a workflow before a running job is removed from the system. These are similar to nodes in a workflow, but do not appear in the standard flow of a workflow. A workflow may have both end and error handlers.

No matter what causes a workflow to end, whether it completes normally or with an error, the end handlers (if one is declared) are executed in the sequence they are defined. In addition, if error handlers are declared, and an exception is raised during workflow processing, the error handlers are executed in the sequence they are defined before the end handlers are executed.

- **Error handlers** are invoked when a workflow process causes an exception and ends the workflow.
- **End handlers** are invoked when the workflow finishes, typically to release resources the workflow might have acquired.

Handlers do not appear as icons in the Workflow Designers Drawing View, since they do not happen with the general flow of the workflow. The error and end handlers are displayed in the Workflow Designers Handler View as two sequence of icons.

It is possible to define a sequence of error handlers to be executed and the same for end handlers, but the intention is not to move the workflow logic from the workflow to the handler part. The handler must only be used for typically end activities and it is not possible to make any loops or branches.

The general structure of a handler declaration is similar to a node declaration, but it has some distinctive features:

```
<Error-Handler>
  <Class-Name>
    com.hp.ov.activator.mwfm.component.builtin.ReleaseResourceHandler
  </Class-Name>
  <Param name="variable0" value="dbServer"/>
  <Param name="variable1" value="ipaddr"/>
</Error-Handler>
```

Notice that the <Class-Name> and <Param> declarations do not appear inside of an <Action> declaration.

## Conventions for Node and Handler Parameters

All workflow nodes and handlers are configured to perform some specific behavior by setting their parameters. A parameter consists of a name and a value. Each node has a unique set of parameters that it responds to, and the node interprets those parameters appropriately to the behavior that it encapsulates.

The following list describes some of the different conventions used that allow nodes to make use of these parameter name/value pairs:

- Some parameters always take a constant value. For example, the `ExecSQLQuery` node takes a parameter that indicates the query which should be performed against the database. The parameter name is `query`.
- Some parameters indicate the name of a case-packet variable that the node should use for a specific purpose. For example, the `ReadFile` node takes a parameter with the name `destination` and a value that is the name of a case-packet variable in which the node should store the contents of the file that it reads.
- Some parameters allow either a variable name or a constant value. In this case, a special syntax is needed to distinguish between the two. The parameter typically has a default, but the alternative must be indicated by a prefix. Some parameters expect a variable name; if a constant is to be specified, the value is preceded by the “constant:” prefix. Some parameters expect a constant value; if a variable name is to be specified, the value is preceded by the “variable:” prefix.

For example, the `Sleep` node has a parameter to indicate the amount of time to sleep. The name of this parameter is `time`. The node expects the value to indicate the name of a case-packet variable that contains the number of milliseconds to sleep, but the value can also be specified as `constant:10000` to indicate that the node should sleep for 10 seconds.

---

### NOTE

---

The node descriptor files indicate the convention expected for each parameter; thus, the Workflow Designer will usually take care to prepend the appropriate prefix.

- Some parameters are Boolean in nature, meaning that the value of the parameter is either “true” or “false.” For example, the `ExecuteExternal` node takes a parameter with the name `wait` and a value of “true” to indicate the node should wait until the process ends before the workflow proceeds to the next node.
- Some parameters are processed as a group with a related purpose. In this case, the parameter names start with a fixed string and have a number appended. For example, the `Add` node takes a list of parameters that are added together. In this case, the parameter names start with the string `op` (for operand). Any number of parameters can be specified, beginning with `op0` and incrementing from there (`op0`, `op1`, `op2`, and so on). The actual order that these parameters appear in the node definition does not matter, they are processed in their numeric order.
- Some parameters use a convention that is a reversal of the previous conventions. In this case, the parameter name is not fixed, but rather is the name of a case-packet variable. The value of the parameter indicates how to set the value of the specified case-packet variable. For example, the `ExecSQLQuery` node performs an SQL query and can set the value of case-packet variables according to the fetched columns. In this case a parameter name might be specified as `customer_name` and the parameter value might be `col0`. This indicates that the case-packet variable `customer_name` should get the value of the first column fetched from the query.

## Case-Packet Variables

Every workflow has a case-packet. A case-packet is a collection of variables that are global to the entire workflow, meaning that each node within the workflow has access to the complete case-packet. The nodes can get and set the value of any case-packet variable.

Each variable in the case-packet has a type. The types supported are:

```
String
Boolean
Integer - (internally a Long value)
Float - (internally a Double value)
Object - (for example, java bean, Array, Map)
```

Most of the nodes provided with Service Activator can only interact with variables of type `String`, `Boolean`, `Integer`, and `Float`. A few nodes can operate on variables of type `Object`. The nodes `ReserveResource`, `ReleaseResource`, and `QueryInventory` treat the Objects as JavaBeans; that is, they must be classes that adhere to the conventions for JavaBeans. Other Service Activator nodes that can support the variable of type `Object` are `UpdateServiceInstance`, `QueryServiceInstance`, `QueryServiceInstanceAll`, `MatchDBQuery`, and `MatchDBStore`. These nodes are not aware of the actual type of variable used because they merely serialize and unserialize the variables to store them to and read them from a database. If you use the variable type `Object` in your custom nodes, carefully determine the actual type of the value of the variable and use it appropriately.

Case-packet variable declarations appear in the `<Case-Packet>` section of a workflow. Each variable declaration specifies the name and type of the variable. It looks like this:

```
<Variable name="service" type="String" />
```

---

### NOTE

By convention your case-packet variables should not be named in all capital letters, since this is the convention that is used for the standard variables that are automatically included in the case-packet (see “Default Case-Packet Variables” on page 36).

---

## Initial Case-Packet Values

Each case-packet variable is initialized to a specific value. This value may be a default value, or it may be explicitly stated in the workflow specification. The default values are:

```
String an empty string
Boolean false
Integer 0
Float 0.0
Object null
```

You can explicitly specify an initial value for any case-packet variable (except for those of type `Object`). These declarations are in the `<Initial-Case-Packet>` section that immediately follows the `<Case-Packet>` section. Each specification of an initial value looks like this:

```
<Variable-Value name="stdNewUserPassword" value="ChangeMe" />
```

## Workflow Contract

A Workflow Contract is used to define input and output parameters for a given workflow. Input parameters consist of zero or more mandatory parameters followed by zero or more optional parameters.

If a workflow is started without providing all the mandatory input parameters an exception is thrown and the workflow is not started.

Only the defined output parameters will be returned when in the returned hashmap.

## Default Case-Packet Variables

Every workflow has a set of standard variables that are automatically included in the case-packet, whether they are declared or not. This table lists the set of default variables for workflows. Notice that all of these variables are, by convention, specified in all capital letters.

**Table 2-1** Default Case-Packet Variables

Name	Type	Description
<i>RET_VALUE</i>	Integer	RET_VALUE is a default case-packet variable, which is updated after each node execution. Each node has an internal attribute called <i>ret_value</i> , which can be changed at runtime. After node execution, the Workflow Manager automatically updates the system case-packet variable RET_VALUE accordingly. Because RET_VALUE is set after each node execution, a check of the node success or failure must be done in the next node. If the RET_VALUE has the value 0, then the node was executed correctly. If the value is -1 then an internal exception was handled by the framework. Any other value is node specific. It is highly recommended to use the WasPreviousNodeOk node to verify whether or not a node was successful.
<i>RET_TEXT</i>	String	If the RET_VALUE system case-packet variable is not equal to 0, then the RET_TEXT system case-packet variable holds a description of the failure or the exception returned from the previous node. As for the RET_VALUE, all nodes have an internal attribute <i>ret_text</i> , which controls the content of the variable. It is possible to use the WasPreviousNodeOk node to save this information in another case-packet variable for later use or presentation.
<i>EX_STEP_NAME</i>	String	After execution of a workflow node the EX_STEP_NAME case-packet variable is set in case the node execution fails. The value is set to the step name of the node which failed. The information can then be used in an error handler to better determine what to do.

**Table 2-1 Default Case-Packet Variables (Continued)**

Name	Type	Description
<i>JOB_ID</i>	Integer	Unique identifier for each running job. You cannot modify this value from within a node.
<i>STEP_NAME</i>	String	Unique name of the node currently being processed. You cannot modify this value from within a node.
<i>HOST_NAME</i>	String	The name of the host where the job is running. You cannot modify this value from within a node.
<i>WORKFLOW_NAME</i>	String	The name of the workflow the job is running. You cannot modify this value from within a node.
<i>WORKFLOW_VERSION</i>	Integer	The workflow version. You cannot modify this value from within a node.
<i>SOLUTION_NAME</i>	String	The Solution Name of the workflow. You cannot modify this value from within a node.
<i>SUBSTEP</i>	String	<p>This variable is not suitable for access by workflow writers. It is useful in rare cases by writers of custom workflow nodes. Workflow nodes can use this to record an indication about partial processing that they have done for a node prior to full completion of the node. This variable is accessed via the <code>WFContext.getSubstep()</code> and <code>WFContext.setSubstep()</code> methods. The setter method ensures that the case-packet gets persisted when this value is set.</p> <p>The Activate node and the AskFor node are the only nodes that currently use this variable. They use this to resume a workflow safely. If an activation is in the middle of execution when the workflow engine is killed (not safely), the workflow engine will resume the workflow and try to re-execute the current node. This would mean that the activation is tried again but this could be catastrophic if the activation is partially completed. To avoid this the Activate node uses the <code>SUBSTEP</code> variable to record the fact that the activation has actually been initiated. If the node is executed again and the <code>SUBSTEP</code> indicates this, the activation will not be retried, and the node will fail (activation_major_code=1, activation_minor_code=2...that is, ERROR/INCONSISTENT). For the AskFor node the input parameters provided to the workflow are saved before the node continues its work.</p>

**Table 2-1**      **Default Case-Packet Variables (Continued)**

<b>Name</b>	<b>Type</b>	<b>Description</b>
<i>START_TIME</i>	Integer	Universal time coordinates (UTC) time when the flow began. This value is maintained in the case of failures and cannot be modified from within a node.
<i>TIMEOUT</i>	Boolean	If true, indicates that a timeout occurred in a previous node (such as AskFor) while waiting for interaction.
<i>PRIORITY</i>	Integer	The priority with which workflows are processed is based on the value of this variable when the PriorityEngineQueue or WeightedEngineQueue is configured in the WorkManagerModule (page 447). This variable is used by e.g. the ActivationModule (page 363) to determine the order in which items on the queue are processed.
<i>DEFAULT_ROLE</i>	String	The workflow's default role.
<i>KILL_ROLE</i>	String	The workflow's kill role.
<i>TRACE_ROLE</i>	String	The workflow's trace role
<i>START_ROLE</i>	String	The workflow's start role

**Table 2-1 Default Case-Packet Variables (Continued)**

Name	Type	Description
<i>STATUS</i>	String	<p>Indicates the state of the workflow. This state variable is set internally by the Workflow Manager, so it cannot be changed by nodes. Valid status for a workflow includes:</p> <p>Initted. The workflow has just been started.</p> <p>Recovered. The workflow has just been recovered after a system shut down.</p> <p>Running. The workflow is currently running code within a node.</p> <p>Transit. The workflow is moving from one node to the next one.</p> <p>Handling Error. An unexpected, uncaught exception was thrown from within a workflow node, and the error handler is currently running.</p> <p>Waiting. The workflow is blocked in a node waiting for some input from a request queue.</p> <p>Awakened. The workflow has just been awakened, unblocking it from the queue on which it was waiting.</p> <p>Finishing. The end handler for this workflow is currently being run.</p>
<i>ETC</i>	String	The full path to the \$ACTIVATOR_ETC directory.
<i>VAR</i>	String	The full path to the \$ACTIVATOR_VAR directory.
<i>FILE_URL_PREFIX</i>	String	This variable can be used to in conjunction with other system case-packet variables to construct a system independent file url. On unix the value is file:// and on windows file:///.
<i>SOLUTION_ETC</i>	String	The full path to the solution/etc directory.
<i>SOLUTION_VAR</i>	String	The full path to the solution/var directory.
<i>UNIQUE_WORKFLOW</i>	Integer	Indicateds if the workflow is unique or not.

**Table 2-1**      **Default Case-Packet Variables (Continued)**

Name	Type	Description
<i>SERVICE_ID</i>	String	The Service Id of the workflow. This variable can be used to correlate a number of jobs to the same service. This variable must be set by the workflow to get a value. Per default it is empty. The variable is shown on the UI in a number of views e.g. the job view.
WORKFLOW_ORDER_ID	String	The Order Id of the workflow. This variable can be used to correlate a number of jobs to the same order id. This variable must be set by the workflow to get a value. Per default it is empty. The variable is shown on the UI in a number of views e.g. the job view.
WORKFLOW_TYPE	String	The workflow type. This variable can be used to indicate which type of workflow it is. This variable must be set by the workflow to get a value. Per default it is empty. The variable is shown on the UI in a number of views e.g. the job view
WORKFLOW_STATE	String	The workflow state. This variable can be used to indicate in which state the workflow is. This variable must be set by the workflow to get a value. Per default it is empty. The variable is shown on the UI in a number of views e.g. the job view.
<i>SCHEDULED_INFO</i>	Object	Used to save scheduled information.
THROW_EXCEPTION	Boolean	This will set the behaviour for all the nodes in the workflow if the throw_except parameter is not set at the node level. Default value is true.
<i>SCHEDULED_INFO</i>	Object	Used to save scheduled information.
BREAK_POINT	String	Used to indicate where a break point is set if running in debug mode
EMPTY_STRING	String	Can be used by SI to assign a case-packet variable to an empty string.
FILE_URL_PREFIX	String	Can be used by SI when it is necessary to provide full url path to a string. By using this it is possible to make the workflow platform independent.
NULL	Object	Can be used by SI to assign a case-packet variable to a null value



**Table 2-2 Special Case Packet Variables**

Name	Type	Description
<i>activation_major_code</i>	Integer	Required for use by the Activate node. After the completion of an activation the Activate node will set this variable to hold the <i>major_code</i> returned by the ExecutionDescriptor from the activation.
<i>activation_minor_code</i>	Integer	Optional for use by the Activate node. After the completion of an activation, if this variable exists in the case-packet, the Activate node will set this variable to hold the <i>minor_code</i> returned by the ExecutionDescriptor from the activation.
<i>activation_stdout</i>	String	Optional for use by the Activate node. After the completion of an activation, if this variable exists in the case-packet, the Activate node will set this variable to hold the <i>stdout</i> returned by the ExecutionDescriptor from the activation.
<i>activation_stderr</i>	String	Optional for use by the Activate node. After the completion of an activation, if this variable exists in the case-packet, the Activate node will set this variable to hold the <i>stderr</i> returned by the ExecutionDescriptor from the activation.
<i>activation_description</i>	String	Optional for use by the Activate node. After the completion of an activation, if this variable exists in the case-packet, the Activate node will set this variable to hold the <i>description</i> returned by the ExecutionDescriptor from the activation.
<i>skip_activation</i>	Boolean	Optional for use by the Activate node. The Activate node will first determine whether this variable exists. If it exists and has a “true” value, the Activate node will not actually perform the activation. It will instead set the <i>activation_major_code</i> to either 0 or the current value of <i>skip_activation_major_code</i> (if it exists), and it will set the <i>activation_minor_code</i> to either 0 or the current value of <i>skip_activation_minor_code</i> (if it exists).

**Table 2-2 Special Case Packet Variables (Continued)**

Name	Type	Description
<i>skip_activation_major_code</i>	Integer	Optional for use by the Activate node. The Activate node will use this to set the <i>activation_major_code</i> if the value of <i>skip_activation</i> is “true”.
<i>skip_activation_minor_code</i>	Integer	Optional for use by the Activate node. The Activate node will use this to set the <i>activation_minor_code</i> if the value of <i>skip_activation</i> is “true”.
<i>RUNTIME</i>	Object	<p>Optional variable containing a Map.</p> <p>If this variable exists, the workflow engine will update it to include details about the execution of each node in the workflow. For most nodes, it only records the timestamp of when the node executed. Some nodes record additional information, for example the AskFor node and the Activate node.</p> <p>The variable is actually a Map of Maps. The first Map is keyed by the step name. The Map for each step contains another map that is keyed by special identifiers such as “timestamp”. These keys are noted in any node description that records additional information in the <i>RUNTIME</i> variable.</p> <p>Here is an example of accessing both the timestamp from the execution of the <i>ConfirmActivationDetails</i> node and the user who did the interaction:</p> <pre>RUNTIME{'ConfirmActivationDetails'}{'timestamp'}</pre> <pre>RUNTIME{'ConfirmActivationDetails'}{'username'}</pre>
<i>RESERVATIONS</i>	Object	<p>Optional variable containing an Array of Java Beans.</p> <p>The <i>ReserveResource</i>, <i>ReleaseResource</i>, <i>ConfirmResourceReservation</i> nodes and <i>ReleaseResourceHandler</i> make use of this variable to record resources that have been reserved. See details in the description of these components.</p>

**Table 2-2 Special Case Packet Variables (Continued)**

Name	Type	Description
<i>message_url</i>	String	<p>Optional variable containing the name of a file or database message id passed from the SocketListenerModule.</p> <p>The SocketListenerModule sets this variable to indicate the name of the file or message id. This tell the workflow from where the message received from the socket is saved. The variable is required for the module to start a workflow. The module will get an error if the variable is not part of the case-packet.</p>
<i>module_name</i>	String	<p>Contains information about who have created the message in the database. This case packet is used together with the message_url when starting a job.</p>

## References to Complex Data Types in Workflow Node Parameters

Many node parameters accept a case-packet variable for the value. Normally you would simply refer to the name of the case-packet variable that contains the value. However, in some cases the case-packet variable is of a more complex data type than one of the simple base types (String, Integer, Float, Boolean), such as an Array or an object that adheres to the JavaBean specification. There are special syntaxes for obtaining values from such objects.

Some workflow nodes will set a case-packet variable of type Object to a complex data type such as an Array. For example, the `XMLMapper` node supports the ability to find multiple tags of the same name, in which case it will save all of the values of those tags into an Array. Similarly, the `QueryInventory` node will return an Object that is a JavaBean or as an array of JavaBeans.

Subsequent nodes in the workflow can refer to these complex data types using special syntaxes appropriate to the data type.

---

### NOTE

These syntaxes are only supported when fetching the value of a case-packet variable, not when setting the value.

---

### Arrays or Vectors

If the case-packet variable contains an Array or a Vector, you may use one of the following syntaxes:

*variable#*      Indicates the number of elements in the array or vector. This is useful for looping through all of the elements of arrays and vectors.

*variable[n]* Indicates the *n*th element of the array or vector. In this case, *n* may be a constant integer value, or the name of another case-packet variable of type Integer. If the array or vector does not contain as many elements as indicated by the index, an exception will be thrown.

### Collections

If the case-packet variable contains a Collection, you may use the following syntax:

*variable#* Indicates the number of elements in the Collection.

### JavaBeans

If the case-packet variable contains an object that adheres to the JavaBean specification, you may use the following syntax to refer to a property of the bean:

*variable.property* This will cause the method *variable.getProperty()* to be invoked to get the value of the property. If the object does not have a method by that name, an exception will be thrown.

### Maps

If a workflow node sets a case-packet variable to contain a Map (a HashMap, for example), the elements of that map can be accessed from the `WFContext.getAttribute()` method using the following syntax:

- `mapVarName{ "key" }` index by a constant key value
- `mapVarName{ keyvar }` index by the value in another case-packet
- `mapVarName#` indicates the number of elements in the map

The syntax supported for all complex data type access in the Workflow Manager is fully recursive. The examples shown in Table 2-3 demonstrate this capability.

---

**NOTE** If the # operator is applied to a null object the result is 0 (zero).

**Table 2-3 Examples of Complex Data Type Access**

Description	Syntax
Arrays indexed by constants	<code>arrVarName[0]</code>
Arrays indexed by variables	<code>arrVarName[intVar]</code>
Length of an array	<code>arrVarName#</code>
Arrays of arrays	<code>arrVarName[0][1]</code>
Fields of beans	<code>beanVar.field</code>
Arrays of beans	<code>arrVarName[0].field</code>
Elements of maps of maps	<code>RUNTIME{ "askForSwitchName" } { "timestamp" }</code>

**Table 2-3 Examples of Complex Data Type Access (Continued)**

Description	Syntax
Fields of beans that are maps	<code>beanVar.field{"key1"}</code>
Arrays indexed by bean fields	<code>arrVarName[beanVarName.field]</code>
Vectors indexed by constants	<code>vecVarName[4]</code>
Vectors indexed by variables	<code>vecVarName[intVar]</code>
Size of a vector	<code>vecVarName#</code>
Number of elements in a HashMap (Map)	<code>hashVarName#</code>
Number of elements in a List (Collection)	<code>listVarName#</code>
Complex expressions	<code>mapVarName{beanVar.field[intVar]} {arrVarName[3]}.field#</code>
	<code>vecVarName[3][intVar]["key2"]</code>

---

**NOTE** The `MapData` node offers an alternate method of extracting data from a `HashMap`. See “`MapData`” on page 206 for additional information.

---

## Queues

Various workflow nodes post messages or requests for interaction onto queues. The term queue is perhaps a misnomer, since there is not any implied ordering of items in the queue.

There are two types of queues:

- **Message queues** are for messages that do not require (or allow) any response. The message is a simple string. The message is placed on the queue and the workflow proceeds without waiting. The workflow node that typically posts messages to a message queue is `PutMessage`.
- **Request queues** are for input requests made by a workflow. There is an external API by which someone may respond to the requests on a request queue. Thus, the request may be satisfied by an operator at the UI or by any other external program. Additionally, the request may be satisfied by another workflow. The workflow node that typically posts requests to a request queue is `AskFor`.

One special feature of requests is that they may have a time-out specified. If the request is not satisfied within the time-out period, the request will be removed from the queue, and the workflow proceeds but with a special flag set to indicate that the preceding request timed out (see “*TIMEOUT*” in Table 2-1 on page 36).

## Queue Names

By default there are no predefined queues. A queue is created simply by the creation of a message or request. Also, a queue is normally removed as soon as the queue becomes empty. However, both of these behaviors may be overridden in the workflow manager configuration.

- Permanent queues may be declared. These queues will never be removed, even if they become empty. Moreover you may declare roles for these permanent queues to indicate who should be allowed to see the queue, even if the queue does not contain items for that user. Use the tag `<Permanent-Queue>` to declare each of these queues.

---

### NOTE

Any role assigned to a permanent queue only affects whether the user should see the queue in the case where there are no messages for the user. This role setting does not grant users the right to see messages that they would not otherwise be able to see.

- Use the configuration parameter `<Queue-Timeout-Seconds>` to indicate that nonpermanent queues should exist for a finite time after they become empty. This is useful in the case that some queues are frequently empty for a short period of time. Without such a time-out value, the operator user interface might not show the queue and then the user might not see a new message on the queue when they are expecting to see it.

## Queues and Roles

Roles are associated with messages and requests, not queues. A queue is *not* tied to a specific role. When an item is placed on a queue, the workflow indicates what role the user must have to see the message or request. The role is set by virtue of the `DefaultRole` set in the workflow, or by having a specific role set on the node.

## Advanced Workflow Techniques

It is not always obvious how to accomplish certain behaviors using workflows. This section provides some example workflows to illustrate useful techniques.

### Spawning Child Workflows

This example consists of two workflows. It illustrates how one workflow (the parent) can start a second workflow (the child) to accomplish a specific task. The parent waits for the child to complete.

This is a typical arrangement. The child workflow may be written to accomplish a specific activation task given a set of input values. The parent workflow may be written to gather the data from a specific input source. Either the parent or the child could be replaced with a new implementation without changing the other.

In this simple example, the parent workflow gathers some data from the operator. It then passes this data to the child workflow and waits for the child to complete. The child performs its actions (in this case, it simply adds the two values together) and returns the result to the parent. The parent then displays the value to the operator.

#### Example 2-1

#### parent

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE Workflow SYSTEM "workflow.dtd">

<Workflow>
  <Name>parent</Name>
  <Description>
    An example parent workflow that gathers some input, spawns a child to
    process it, waits for the child to complete, issues a message with the
    result of the operation.
  </Description>
  <Start-Node>Gather Input</Start-Node>

  <Nodes>
    <Process-Node>
      <Name>Gather Input</Name>
      <Description>Asks for the two operands</Description>
      <Action>
        <Class-Name>com.hp.ov.activator.mwfm.component.builtin.AskFor</Class-Name>
        <Param name="variable0" value="operand1" />
        <Param name="variable1" value="operand2" />
        <Param name="queue" value="common_queue" />
      </Action>
      <Next-Node>Start work</Next-Node>
    </Process-Node>

    <Process-Node>
      <Name>Start work</Name>
      <Description>Starts another workflow to do the summation</Description>
      <Action>
        <Class-Name>
          com.hp.ov.activator.mwfm.component.builtin.StartJobAndWait
        </Class-Name>
        <Param name="workflow_name" value="constant:child"/>
        <Param name="variable0" value="operand1" />
      </Action>
    </Process-Node>
  </Nodes>
</Workflow>
```

```
<Param name="variable1" value="operand2" />
<Param name="variable2" value="JOB_ID"/>
<Param name="destination2" value="parent_job_id"/>
<Param name="queue" value="sync" />
<Param name="outputvar0" value="sum" />
<Param name="outputvar1" value="status" />
</Action>
<Next-Node>Show result</Next-Node>
</Process-Node>

<Process-Node>
  <Name>Show result</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PutMessage
    </Class-Name>
    <Param name="message" value="Status %s Sum %s" />
    <Param name="param0" value="status"/>
    <Param name="param1" value="sum"/>
    <Param name="queue" value="info" />
  </Action>
</Process-Node>

</Nodes>

<Case-Packet>
  <Variable name="operand1" type="Integer"/>
  <Variable name="operand2" type="Integer"/>
  <Variable name="status" type="String"/>
  <Variable name="sum" type="Integer"/>
</Case-Packet>

<Initial-Case-Packet>
  <Variable-Value name="operand1" value="10"/>
  <Variable-Value name="operand2" value="5"/>
  <Variable-Value name="status" value="unknown"/>
</Initial-Case-Packet>
</Workflow>
```



## Example 2-2 child

This trivial child workflow is not attempting to demonstrate the tasks that can be accomplished in a workflow, but rather only trying to elucidate a proper technique for synchronizing between a parent and child workflow.

The important point to note about the child workflow is its use of the `SyncHandler`. Since the parent workflow is waiting for the child, we need to be sure that no matter what causes the child workflow to end, the parent gets notified that the child is complete. If there is some error in processing the child workflow, or if it gets terminated in an unexpected fashion, then the regular flow of the child will be interrupted. Thus, we use an end handler to send the result back to the waiting parent. Then, regardless of what causes the child to end, the parent will get a notification.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Workflow SYSTEM "workflow.dtd">

<Workflow>
  <Name>child</Name>
  <Start-Node>Add</Start-Node>

  <Nodes>
    <Process-Node disablePersistence="true">
      <Name>Add</Name>
      <Description></Description>
      <Action>
        <Class-Name>
          com.hp.ov.activator.mwfm.component.builtin.Add
        </Class-Name>
        <Param name="op1" value="operand2"/>
        <Param name="op0" value="operand1"/>
      </Action>
      <Next-Node>VariableMapper</Next-Node>
    </Process-Node>

    <Process-Node>
      <Name>VariableMapper</Name>
      <Description></Description>
      <Action>
        <Class-Name>
          com.hp.ov.activator.mwfm.component.builtin.VariableMapper
        </Class-Name>
        <Param name="status" value="done"/>
      </Action>
    </Process-Node>
  </Nodes>

  <End-Handler>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.SyncHandler
    </Class-Name>
    <Param name="job_id" value="parent_job_id"/>
    <Param name="variable0" value="operand1"/>
    <Param name="queue" value="constant:sync"/>
    <Param name="destination0" value="sum"/>
    <Param name="variable1" value="status"/>
  </End-Handler>

  <Case-Packet>
    <Variable name="JOB_ID" type="Integer"/>
    <Variable name="controller_job_id" type="Integer"/>
    <Variable name="operand1" type="Integer"/>
  </Case-Packet>
</Workflow>
```

```
        <Variable name="operand2" type="Integer"/>
        <Variable name="status" type="String"/>
    </Case-Packet>
    <Initial-Case-Packet>
        <Variable-Value name="status" value="error"/>
    </Initial-Case-Packet>
</Workflow>
```

## Using Timeouts

When your workflow is waiting for external input (from a user or another executable), you may want to ensure that the workflow does not wait indefinitely. The way to accomplish this is with a timeout. The AskFor node provides this capability.

In this example, the workflow starts and then pauses to allow the user to enter a text. If the user does not do this within ten seconds, a timeout message is sent. Otherwise, a message including the user's string is sent.

Notice how the workflow checks the value of the variable TIMEOUT to see whether a timeout has occurred.

### Example 2-3 timeout

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Workflow SYSTEM "workflow.dtd">

<Workflow>
  <Name>timeout</Name>
  <Description>Small timeout test</Description>
  <Start-Node>Request string</Start-Node>
  <Nodes>
    <Process-Node>
      <Name>Request string</Name>
      <Description>Request the string from the user</Description>
      <Action>
        <Class-Name>
          com.hp.ov.activator.mwfm.component.builtin.AskFor
        </Class-Name>
        <Param name="variable0" value="your_string" />
        <Param name="queue" value="operator" />
        <Param name="timeout" value="10000" />
      </Action>
      <Next-Node>Was there a timeout?</Next-Node>
    </Process-Node>

    <Rule-Node disablePersistence="true">
      <Name>Was there a timeout?</Name>
      <Description>Checks state of TIMEOUT variable</Description>
      <Action>
        <Class-Name>
          com.hp.ov.activator.mwfm.component.builtin.Equal
        </Class-Name>
        <Param name="op1" value="TIMEOUT" />
        <Param name="op2" value="constant:true" />
      </Action>
      <True-Next-Node>Yes timeout</True-Next-Node>
      <False-Next-Node>No timeout</False-Next-Node>
    </Rule-Node>

    <Process-Node>
      <Name>Yes timeout</Name>
      <Description>Shows timeout message</Description>
      <Action>
        <Class-Name>
          com.hp.ov.activator.mwfm.component.builtin.PutMessage
        </Class-Name>
        <Param name="message" value="No input was received before the timeout." />
        <Param name="queue" value="operator" />
      </Action>
    </Process-Node>
  </Nodes>
</Workflow>
```

```
<Process-Node>
  <Name>No timeout</Name>
  <Description>Shows string</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PutMessage
    </Class-Name>
    <Param name="message" value="Your string was: '%s'" />
    <Param name="param0" value="your_string" />
    <Param name="queue" value="operator" />
  </Action>
</Process-Node>
</Nodes>

<Case-Packet>
  <Variable name="your_string" type="String" />
</Case-Packet>

<Initial-Case-Packet>
  <Variable-Value name="your_string" value="Hello world!!!" />
</Initial-Case-Packet>
</Workflow>
```

## Using Prioritization in Workflows

You can prioritize the order in which workflows are processed by the Workflow Manager. You can also prioritize activation items.

### Prioritizing Workflow Node Processing

There are multiple threads (the number is configurable) in the Workflow Manager that are dedicated to working on jobs. Each node in a workflow is handled as an independent piece of work. When the job starts, the “start-node” is placed at the end of the work queue. Eventually one of the worker threads picks it up and executes it. When that node completes, the worker thread puts the “next-node” at the end of the work queue, and so on.

The `WorkManagerModule` can be configured to use the `SimpleEngineQueue`, `WeightedEngineQueue`, or the `PriorityEngineQueue`. The `SimpleEngineQueue` simply does not handle prioritization of workflow nodes. The `WeightedEngineQueue` and `PriorityEngineQueue`, however, do.

You can specify the priority of a workflow by using the `PRIORITY` default case-packet variable. Then, if you specify the `PriorityEngineQueue` in the `WorkManagerModule`, nodes from higher priority workflows will be placed in the work queue ahead of nodes from lower priority workflows. If the `PRIORITY` case-packet variable is not found, the priority for all nodes in that workflow is assumed to be 0. Negative priority values are supported. The `WeightedEngineQueue` is very similar to the `PriorityEngineQueue` exception for it makes it prioritization in a weighted way, i.e. if you have high priority jobs then they will be executed first with the exception that sometimes a low priority job will also be executed. By using the `WeightedEngineQueue` you avoid starvation.

### Prioritizing Activation Items

To use the Workflow Manager, you must configure an activation module. One class is supplied for this purpose: **ActivationModule**.

The `ActivationModule` has its own pools of threads (the number is configurable), separate from the engine worker threads, that perform activations, thereby freeing the worker threads to operate on nodes in other workflows *while* the activations are being performed. Thus, it is possible to have hundreds of pending activations while other workflows are being processed. These threads pull activation requests off a queue. Like the `WorkManagerModule`, the `ActivationModule` can be configured to use the `SimpleEngineQueue`, `WeightedEngineQueue`, or the `PriorityEngineQueue`.

The default installation configures the `ActivationModule` but uses the `WeightedEngineQueue`.

### How to Determine Whether Prioritization is Working

Prioritization manifests itself in a number of ways. It only comes into effect when there is a backlog of work. If there are enough threads available to process the work, then prioritization will have no effect.

### Testing Prioritization in Working Threads

Unless you have configured `Min-Threads` differently, there are typically 5 worker threads running in the system for processing workflows. If there are fewer than 6 jobs performing active work—not waiting for input or waiting for an

activation—prioritization will have almost no effect. This is because every time a work item is placed on the queue, a worker thread will be available to work on it regardless of its priority.

Follow these steps to verify that the prioritization of working threads is functioning correctly:

1. Set the value of *Min-Threads* value to 1. This will make a single worker thread available for working on workflows.

---

**NOTE**

*Max-Threads* only causes new worker threads to be spawned if the *Spawn-List-Length* is exceeded.

2. Restart the Service Activator.
3. Write a workflow that will run for a long time (by looping many times, say 1000 loops). Do NOT use a `Sleep` node.
4. Copy that workflow, and rename it.
5. In the second workflow, set the initial value of *PRIORITY* to 1.
6. Start the low-priority workflow first.
7. While the low-priority workflow is running, start the high-priority workflow.
8. The high-priority workflow should finish first.

---

**NOTE**

Remember though, prioritization at this level is not very deterministic. It is still possible for the low-priority jobs to get some processing slices.

### Testing Prioritization of Activation Threads

If you have configured the `ActivationModule`, there are multiple threads available for performing activations (default is *max\_threads=20*). Thus, if 20 workflows reach their `Activate` node, they can all be processed simultaneously. Prioritization will come into effect *only* if more workflows reach their activate node while the 20 threads wait for activations to complete.

Follow these steps to verify that the prioritization of activation threads functions correctly:

1. Configure the `ActivationModule` with both *min\_threads* and *max\_threads* set to 1. Now there is a single activation thread available.
2. Write a workflow that performs a simple activation (that takes a few seconds to complete).
3. Copy that workflow, rename it, and set the initial value of *PRIORITY* to 1.
4. Now start 3 copies of the low-priority workflow followed by 3 copies of the high-priority workflow.
5. You should see the first low-priority workflow complete its activation first, since it would have initially reached the `Activate` node and started its activation before the other workflows were started. You should then see the 3 high-priority workflows complete their activation, followed by the other two low-priority workflows.

**See Also**

- “ActivationModule” on page 363
- “WorkManagerModule” on page 447

## Uploading Data from a Task Activation

Arbitrary (Serializable) data can be uploaded (returned) from a task activation for use by the invoking workflow. An atomic task may need to query a target device and return state information to the workflow, for example.

The data uploading capability is provided by the `PARContext` interface available to a plug-in. See *HP Service Activator—Developing Plug-ins & Compound Tasks* and the *Javadocs* for `PARContext` and `DataUploader` for additional information.

The `Activate` node makes uploaded data available in a case-packet variable stored as a `HashMap`. Data can be extracted from the `HashMap` in one of the two ways: using the workflow syntax for accessing maps or using the `MapData` node.

### See Also

- “Activate” on page 96 for more information about making uploaded data available in a case-packet variable stored as a `HashMap`
- “Maps” on page 44 for more information about using the workflow syntax for accessing maps to extract uploaded data from a `Map`
- “MapData” on page 206 for more information about using the `MapData` node to extract uploaded data from a `Map`



## Deploying Workflows

The workflows will typically be placed under `$ACTIVATOR_ETC/workflows` if Solution Separation is not used. Where the workflows will be placed under the solution directory in case Solution Separation is used. The files require an `.xml` extension so that the Workflow Manager recognizes them as valid workflow files. The Workflow Designer automatically starts to read and puts new workflows into the directory `$ACTIVATOR_ETC/workflows`.

The workflows must be deployed to the system database to be used by the Workflow Manager. This can be done by the Workflow Designer either from the command line or the UI. Then after the workflows are deployed either Service Activator must be restarted or the reload workflow operation must be performed from the Operator Interface.

## Clustering Considerations

When writing workflows you need first of all to consider if the workflow is going to be used in a cluster solution or not. Then next if you are going to use persistence or not. If the answer is yes to both questions then there are a number of things which needs to be considered.

Temporary files are very often used to pass information around in a workflow which is possible if persistence are done at the right places. If this is not considered and the cluster node where the job is running fails the access to the temporary file is lost at the job will continue its execution on an other cluster node. To use temporary files no persistence must be done from the workflow node which generate the file is executed to the end of the use of the temporary file.

An alternative to this would be to save the data to the database which all cluster nodes have access to. Two workflow nodes can be used to read and write data to a message database. The `ReadDataFromDatabase` and `WriteDataToDatabase`. To remove database from the database the node `RemoveData` can be used. When using the nodes the identifier for the data is a message id and it is this identifier which must be passed around in the workflow.

The `SocketListenerModule` can be configured to generate both kind of input and the `SocketSenderModule` is also capable to read the information from both a file and the database.

The `XMLMapper` can be used directly to read data from the database or the file system and by combining this with the `SocketListenerModule` a workflow can be written which will work both when the `SocketListenerModule` is configured to write information to the file system and to the database.

Also the plugins can read data from the database and write data to the database. Two methods exist there two. One for reading data and one for writing data. So it is easy to pass data from or to the plugin by using the message data database and then just send the message id to the plugin as an argument or receive the message id in the upload data object.

---

## **3 Using the Workflow Designer**

The HP Service Activator Workflow Designer is a graphical tool you can use to easily create and edit workflows. Creating workflows requires knowledge about the available workflow nodes and workflow modules.

To retain compatibility across all supported platforms, you must use the forward slash '/' as file path separators.

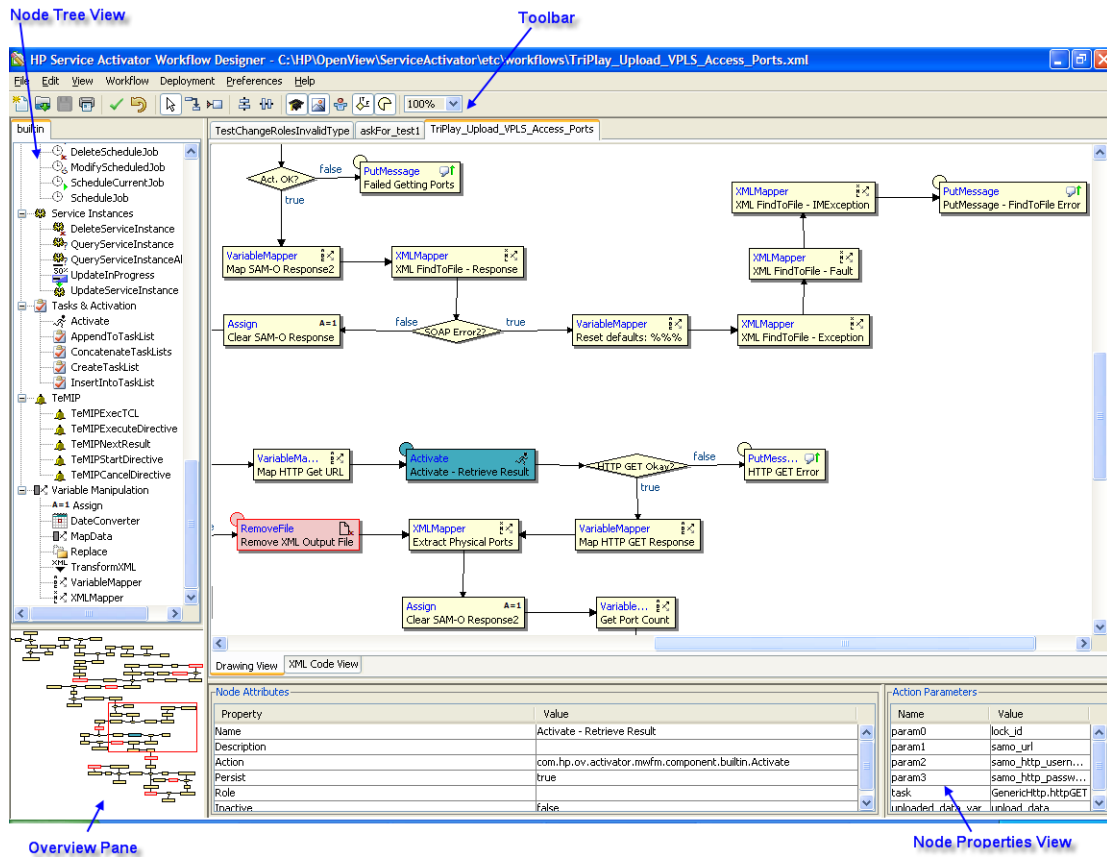
## Navigating the Workflow Designer

This tool allows you to edit workflows graphically. The graphical editor supports all the functionality of a workflow. Its main benefits are found in laying out the workflow so that it can be easily understood, connecting nodes or changing the sequence of a workflow, and setting the parameters of the various workflow nodes.

The Workflow Designer stores workflows in an XML format. A workflow can be edited using the Workflow Designer even if it was created outside of Workflow Designer. In that case, all nodes will have the same position and it is up to the user to place the nodes in proper positions. In addition, the Workflow Designer allows you to view the XML source file in text mode.

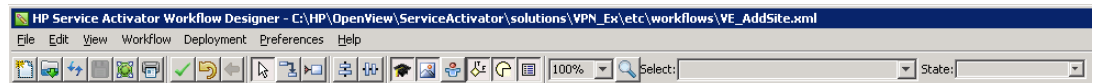
The Workflow Designer graphical editor consists of five sections: the toolbar (at the top), the node tree view (at the top-left), the workflow view (top-right), the overview pane (bottom-left), and the node attributes view (bottom-right). The Workflow Overview and the Node Attributes view can be hidden and displayed as desired.

**Figure 3-1 Workflow Designer**



## Understanding Workflow Designer Features

When you have edited a workflow file but not yet saved it, the file name shown in the title bar is followed by asterisks (\*\*), as shown here:



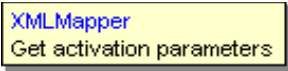
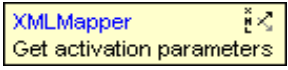
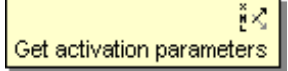
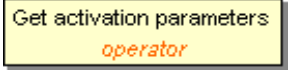
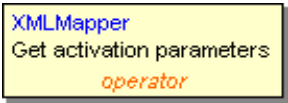
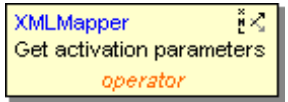
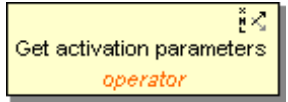
The following table shows the different styles in which process nodes can be displayed:

**Table 3-1 Process Nodes**

Get activation parameters

Basic layout of a process node; the name of the node is always displayed.

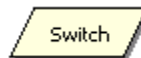
**Table 3-1 Process Nodes**

 <p>XMLMapper Get activation parameters</p>	<p>Process node displaying the node's class name.</p>
 <p>XMLMapper Get activation parameters</p>	<p>Process node displaying the node class name and icon.</p>
 <p>Get activation parameters</p>	<p>Process node displaying the node icon.</p>
 <p>Get activation parameters <i>operator</i></p>	<p>Process node displaying the role.</p>
 <p>XMLMapper Get activation parameters <i>operator</i></p>	<p>Process node displaying the node class name and role.</p>
 <p>XMLMapper Get activation parameters <i>operator</i></p>	<p>Process node displaying the node class name, node icon, and role.</p>
 <p>Get activation parameters <i>operator</i></p>	<p>Process node displaying the node icon and role.</p>

In the workflow view, rule nodes are always displayed using a diamond shaped icon as shown below:



In the workflow view, switch nodes are always displayed using a parallelogram shaped icon as shown below:

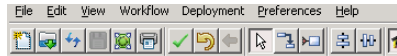


---

## Using the Main Menu

Figure 3-2 shows the main menu of the Workflow Designer.


**Figure 3-2** Workflow Designer Main Menu



The main menu is the only visible part of the application when you launch it the first time (before you load any files). The menu bar at the top of the toolbar consists of the following menus:

### 1. File

The File menu contains the usual commands: create a new workflow, open an existing workflow, save a workflow, save all workflows, save a workflow under a different name (Save As...), print a workflow, close a workflow, and quit the application.

You can also quit the application by clicking the  button on the top right corner window title bar.

The File menu also contains the menu entry `Switch Directory`, which makes it possible to switch the default directory. The default directory is used by the workflow designer when trying to open a workflow.

### 2. Edit

The Edit menu contains commands and actions for editing a workflow. There are four options available:

Undo : undo the last operation.

Copy : copy the currently selected nodes to the clipboard.

Paste : paste the nodes currently in the clipboard into the currently active workflow. This operation can not be used to copy nodes from one workflow to another.

Paste Special : If the nodes in the clipboard were copied from another workflow, you must use this operation to paste them into the currently displayed workflow.

### 3. View

Select this to change certain visual aspects of a workflow. There are six options available:

View Node Class: shows/hides the class name of the process nodes.

View Node Icons: shows/hides the node icon of the process nodes.

View Roles: shows/hides the roles of the process nodes.

True/False Tags: shows/hides the rule nodes tags that mark its true and false branches and show/hides the switch nodes tags that mark the value of the branches.

View Workflow Overview: shows/hides the overview pane. If Workflow Overview is selected the State Overview is hidden.

View State Overview show/hides the overview pane. If State Overview is selected then Workflow Overview would be hidden.

View Node Attributes: shows/hides the note attributes view. The node attributes view shows the attributes of the currently selected nodes in a workflow.

#### 4. Workflow

This menu allows you to change almost every non-visual aspect of the workflow configuration:

##### a. Workflow Settings...

Shows the Workflow Settings dialog, which contains four tabs:

1. General - Allows you to set the name of the workflow and to set an indication of whether this workflow should be automatically started when the Workflow Manager is started, and to set an indication of whether the Workflow Manager should only allow a single instance of this workflow to be running at any one time.

Unique Workflow: Check the Unique Workflow check box to create a unique workflow. If only one instance of workflow exists, it is known as a unique workflow.

Initialize Workflow: Check the Initialize Workflow check box to start a workflow automatically when the workflow engine starts.

Here you can also enable or disable collecting audit and/or statistical records as well as statechange records for the workflow. Uncheck the Enable auto generated audit check box to exclude autogenerated audit messages.

Persist Nodes: Check the Persist Nodes check box to enable persistence of workflow data. Persistence allows you to save workflow data from time to time.

2. Description - Provides a large text area for composing a multi-line description of the workflow.
3. Roles - Allows you to edit the roles that may carry out operations on this workflow.

##### b. Add New Roles...

Goes directly to the Roles tab of the Workflow Properties dialog for adding, deleting, and assigning workflow rules.

##### c. Edit Case-Packet...

Shows the Case-Packet dialog for adding, modifying and deleting case-packet variables, including setting the initial value of variables.

##### d. Workflow Contract...

Goes directly to the Workflow Contract dialog for adding and deleting case-packet variables to the contract.

##### e. Persistency



Allows you to enable or disable persistency. If persistence is enabled, the workflow data is stored in the database.

#### 5. Deployment

Deploying workflows to the database.

##### a. Deploy Current Workflow...

Allows to deploy the workflow on which the user is currently working on.

##### b. Deploy All Open Workflows...

Allows to deploy all the workflows that are opened or exist in the Workflow Designer UI.

##### c. Deploy Workflows...

Allows to select and deploy workflows that exist in the file system.

First time deployment is done the user will be prompt for database user and password and optionally for database instance, port, and host name. The values should match the values provided when running ActivatorConfig, which are the values for the system database. The Workflow Designer will then remember the values as long as it is running.

#### 6. Preferences

##### a. XML Default Directory...

Sets the default directory where the application looks for workflow XML files. The program recalls any change made to this parameter in future uses.

#### 7. Help

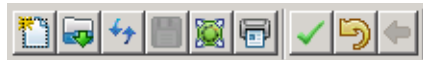
Select this to see information about the Workflow Designer.

---

## Using the Main Utilities Toolbox

The main utilities toolbox consists of the first six buttons on the left side of the toolbar.

**Figure 3-3** Workflow Designer Main Utilities Toolbox



1. New File

Select this to create a new workflow. The application displays a *Workflow Settings* dialog box so that you can assign initial workflow properties.



2. Open File

Select this to open an existing workflow. You can use this to load an already existing workflow or workflows in the *Workflow Designer* window.



3. Save File

Select this to save an open workflow using its current name. To save your file under a different name, choose *File* from the drop-down menu, and then choose *Save As...*



4. Print Workflow

Select this to see the *Print* dialog box and send a graphical representation of the workflow to the specified printer.



5. Check Workflow

Select this to check the validity of the workflow you have designed. The application detects any errors in the workflow and displays a warning message that describes them. The most valuable check performed verifies that all required parameters in the nodes have been assigned a value.



6. Undo

Select this to undo the last action.

---

**NOTE**

The *Undo* operation can only undo delete operations. The undo buffer does not keep a record of other operations such as adding nodes, moving nodes, changing node parameters, and so on.



7. Back

Navigate back to previous workflow.

## Using the Visual Properties Toolbox

When editing a workflow graphically, the Workflow Designer is in one of a few modes. The current mode is indicated by the highlighted button in the toolbox.

By default, the Workflow Designer is in the Select mode, represented by the white pointer icon. Some modes only last for a single operation before the mode reverts to the Select mode. The other buttons in the toolbox do not represent modes. Two of the buttons are used to align nodes vertically or horizontally; you must select at least two nodes in the workflow to operate on for these buttons to have an effect. The last three buttons are toggle buttons used for showing or hiding node class names, node icons, and roles.

**Figure 3-4** Workflow Designer Visual Properties Toolbox



The icons in the visual properties toolbox represent the following modes or functions:



### 1. Select

This represents the default mode of the Workflow Designer. The following functions are available in this mode:

- Add a new node  
Click a node in the node palette; then, click in the design window to create a new node at that location.
- Move a node  
Click a node in the design window, and drag it to a new location. Any arrows connecting it to other nodes are automatically pulled along with it.
- Select/move multiple nodes  
Hold down the control key (**CTRL**) while clicking nodes, or click and drag the cursor to make a box around the nodes you want to select. Each selected node becomes highlighted. At this point you can perform multi-node functions such as moving all of the selected nodes (hold down the **CTRL** key and drag the cursor) or aligning all of the selected nodes by clicking the Horizontal Align icon or the Vertical Align icon in the toolbox.
- Delete  
Click a node in the design window, and press the delete key to remove a node and its connecting arrows.
- Edit node properties  
Double-click a node to bring up the Edit Node Properties dialog. For details on editing node properties, see Using the Edit Node Properties Dialog.
- Context sensitive menu  
Right-click a node to get a context sensitive menu for that node.



### 2. Draw Arrow

Used to draw an arrow between two nodes:

- a. Click the Draw Arrow icon
- b. Click a source node (arrow starting node)
- c. Click an end node (arrow ending node).

If you select a rule node as the starting node for an arrow, the application asks you if this arrow should point to the node associated with `true` or `false`.

If you select a switch node as the starting node for an arrow, the application asks you if this arrow should point to the node associated with `default` or `case0`, `case1`, etc.



3. Set Initial Node

- a. Click the Initial Node icon
- b. Click a node that should become the starting node in the workflow. A red triangle is attached to the node to indicate that it is now the initial node.



4. Vertical Align or Horizontal Align

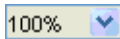
Use these options when you want the arrows connecting the nodes to be perfectly vertical or horizontal.

- a. Select multiple nodes that are to be aligned. You may use any method to select multiple nodes (see “Select” on page 67).
- b. Click the Vertical Align or Horizontal Align icon. All of the selected nodes will be moved to align with the first node selected.



5. Show/hide class name, icon and roles for process nodes

These five toggle buttons are used to change the view mode for the current workflow.



6. Zoom

- a. Click and select one of the available zoom percentages from the drop down list. Node icons are only displayed if the zoom level is set to 100%.

---

**NOTE**

When adding a new node to a workflow by clicking a node from the workflow node tree, the cursor will change to a cross-hair and the leftmost buttons in the Visual Properties Toolbox will become disabled.



7. Search

- a. When clicking this button a pop dialog is shown which makes it possible to enter different search criteria to search for workflow nodes.

---

**NOTE**

When adding a new node to a workflow by clicking a node from the workflow node tree, the cursor will change to a cross-hair and the leftmost buttons in the Visual Properties Toolbox will become disabled.



8. SelectWorkflow Node

- a. Click and select one of the available workflow nodes.



9. SelectWorkflow state

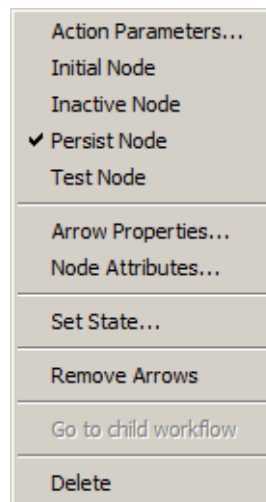
- a. Click and select one of the available workflow states.

---

## Using the Context Sensitive Menu

Right-click a node in the workflow design window to bring up a context sensitive menu.

**Figure 3-5** Context Sensitive Menu



The menu contains the following actions:

1. Action Parameters  
Shows the Edit Node Properties dialog for this node (with the Action Parameters tab active). This is the same dialog that appears if you double-click a node.
2. Initial Node  
Set this node to be the starting node of the workflow.
3. Inactive Node (only for a Process Node)  
Toggles whether this node is active or inactive. When the workflow is executed, the Workflow Manager will ignore any inactive node, skipping to the next node in the workflow (if there is one). Rule and switch nodes do not have this option since the Workflow Manager would not know how to proceed from the node.
4. Test Node  
Toggles whether this is a active or test node. When the workflow is executed, the Workflow Manger will not execute the test node unless the Test-Mode is set to true in the mwfm.xml configuration file.
5. Persist Node  
Toggles whether persistence should be done after execution of the node.
6. Arrow Properties  
Set what kind of arrow comes out of this node. By default all arrows are “Straight”. You may also choose one of the arrows with elbows to make the workflow easier to read.
7. Node Attributes

Shows the Edit Node Properties dialog for this node (with the “Node Attribute” tab active). This is the same dialog that appears if you double-click a node.

8. Remove Arrows

Removes any arrow(s) emanating from this node. You can change the destination of a node simply by clicking the “Connect Nodes” icon from the toolbox and clicking the two nodes to be connected; you must use this menu item to remove an arrow from a node.

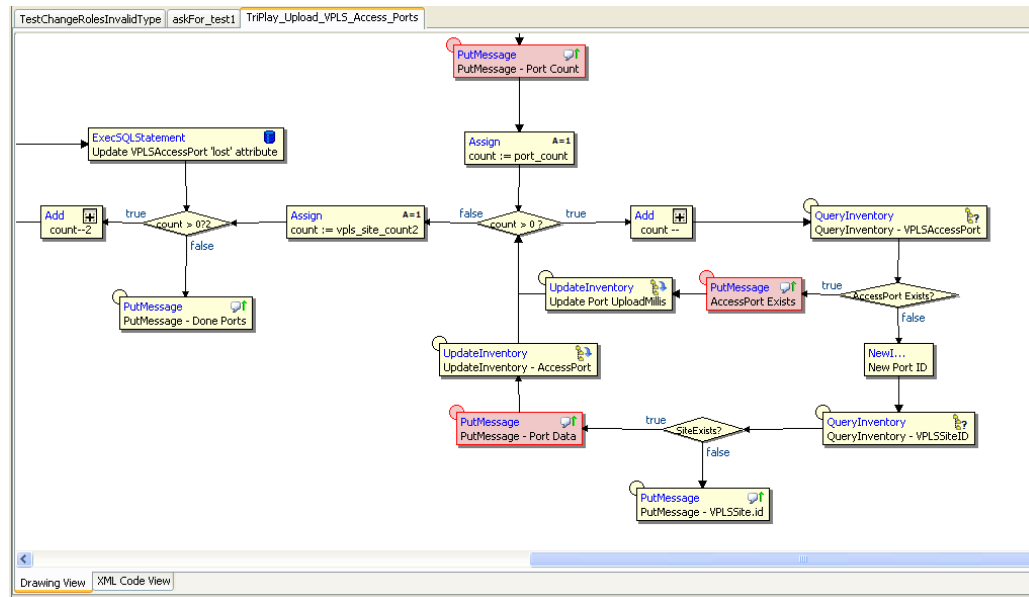
9. Delete

Remove the node from the workflow. You can also select the node and press the **[Delete]** key.

## Using the Workflow Views

The Workflow Designer can display workflows graphically or as XML code (read-only). You can switch between the tree views by clicking the Drawing View tab, Handler View tab, or the XML Code View tab.

**Figure 3-6** Workflow Drawing View

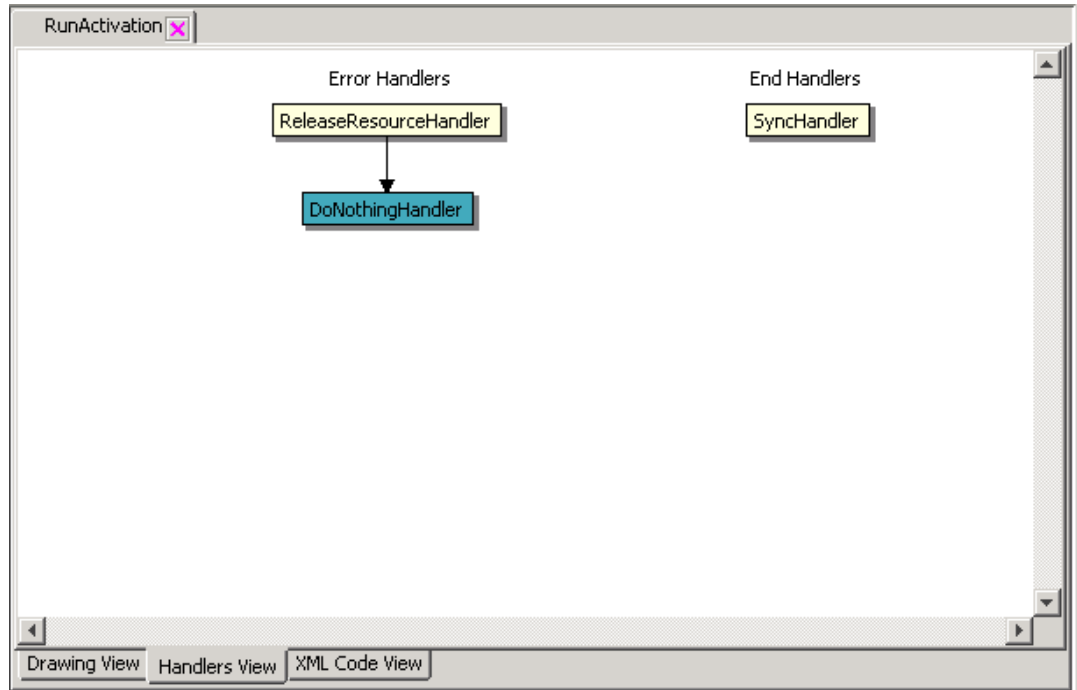


If a single node is selected while switching from Drawing View to XML Code View the Workflow Designer will scroll down to the XML definition of the selected node and the node name will become highlighted. An example of this is shown in Figure 3-7.

If multiple nodes are selected the node that was most recently selected will be highlighted in the XML Code View.

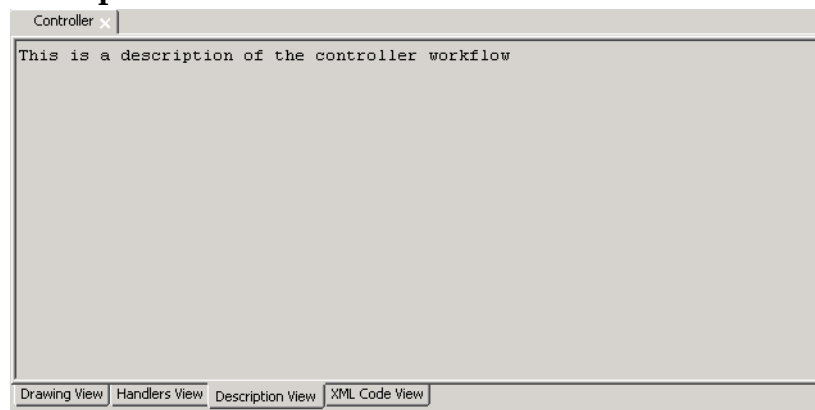
In the Handlers view the end and error handlers which are defined will be shown in two different columns. If no handlers are defined the Handler View will be empty.

**Figure 3-7**      **Handlers View**



In the Description view the overall description of the workflow can be seen. The description field can be edited in the workflow settings.

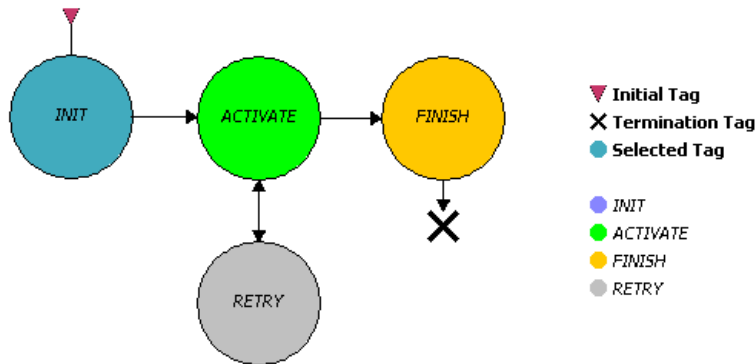
**Figure 3-8**      **Description View**





The state view shows the workflow represented as a (simplified) state diagram.

**Figure 3-9 State View**



The XML Code View show the xml content.

**Figure 3-10 Workflow XML Code View**

```

<Name>L2Data_L2QinQ_reconStats</Name>
<Action>
  <Class-Name>com.hp.ov.activator.mwfm.component.builtin.UpdateInventory</Class-Name>
  <Param name="bean" value="com.hp.ov.activator.inventory.sam.L2QinQIF"/>
  <Param name="key_field0" value="id"/>
  <Param name="key_value0" value="l2if_id"/>
  <Param name="key_field1" value="reconResult"/>
  <Param name="key_value1" value="BOTH"/>
</Action>
<Next-Node>count > 0?</Next-Node>
</Process-Node>
<Process-Node inactive="true">
  <Name>PutMessage - Debug</Name>
  <Action>
    <Class-Name>com.hp.ov.activator.mwfm.component.builtin.PutMessage</Class-Name>
    <Param name="message" value="L2 Q-in-Q interface does not exist - objectFullName = %s"/>
    <Param name="queue" value="recon"/>
    <Param name="param0" value="l2if_object_full_name_array[count]"/>
    <Param name="param1" value="vpls_id"/>
    <Param name="param2" value="current_date"/>
  </Action>
</Process-Node>
  
```

A maximum of 20 workflows can be opened simultaneously. When you reach this limit, you will have to close one or more workflows in order to be able to open new workflows. In addition, you can not open two workflows with identical workflow names; this restriction also applies if their file names are different.

You can switch between the open workflows by clicking the corresponding tabs or by using the **CTRL-q** keyboard shortcut. Additionally, you can close the currently displayed workflow by using the **CTRL-w** keyboard shortcut.

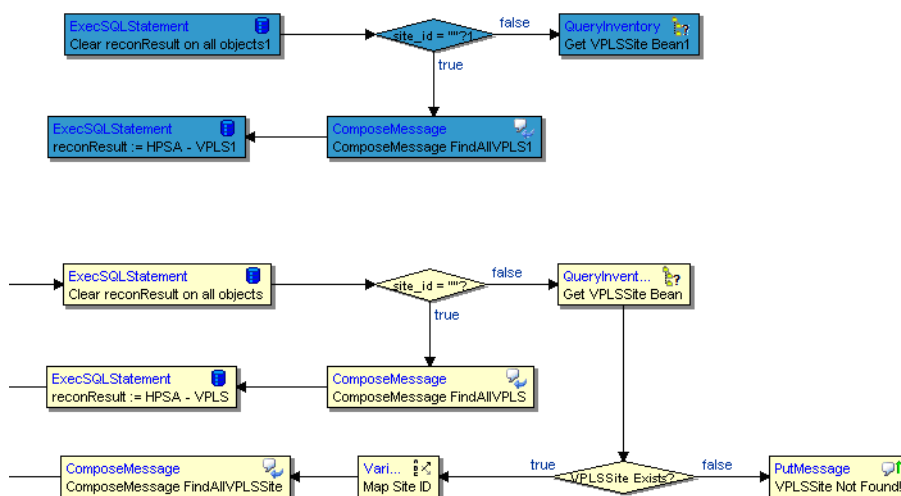
## Copying and Pasting Workflow Nodes

The Workflow Designer has the following copy and paste capabilities:

- Copying and pasting one or more nodes within the same workflow
- Copying and pasting one or more nodes between workflows opened in the same instance of Workflow Designer, the `Edit:Paste Special` menu option must be used for this operation.
- Copying and pasting workflow nodes to another application. For example, Microsoft Notepad.

When you copy and paste multiple nodes the Workflow Designer will preserve all arrows for which the source as well as the destination nodes exists in the copy buffer; other arrows will be deleted. This is illustrated in Figure 3-8 where five nodes were copied and then pasted into the same workflow.

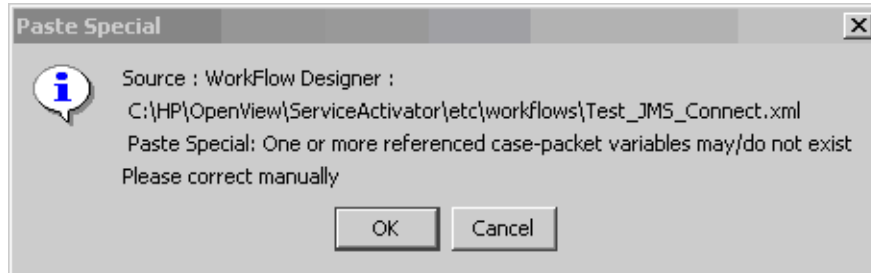
**Figure 3-11** Copy and Paste of Multiple Nodes



If you copy one or more workflow nodes in one workflow and paste them into another workflow by using `Edit:Paste Special`, the Workflow Designer will display a warning dialog; see Figure 3-9. The warning dialog is displayed to make it clear to the user that

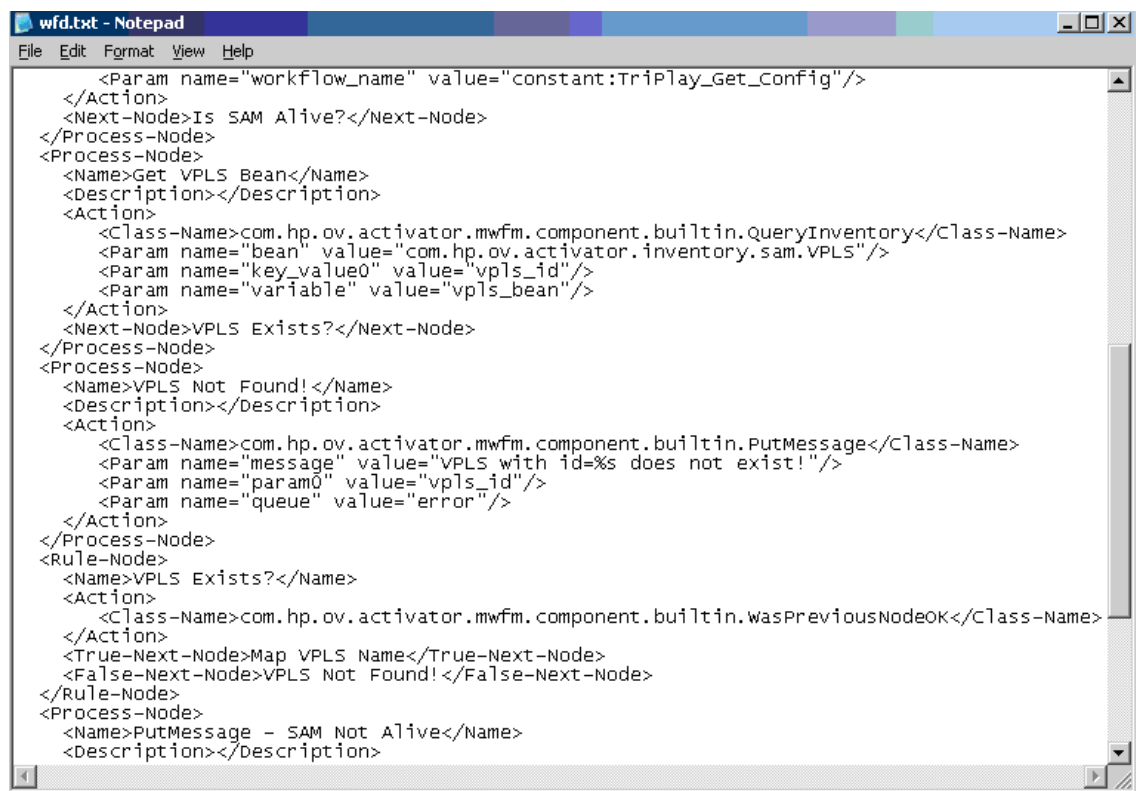
issues may occur if case-packet variables in the pasted nodes are in conflict with case-packet variables already existing in the destination workflow. For example, conflicting types.

**Figure 3-12 Paste Special Warning Dialog**



Finally, it is possible to copy workflow nodes in Workflow Designer and paste them to another application, such as Microsoft Notepad. In that case the workflow nodes will be pasted as formatted XML – see Figure 3-10.

**Figure 3-13 Workflow Nodes Copied to Microsoft Notepad**

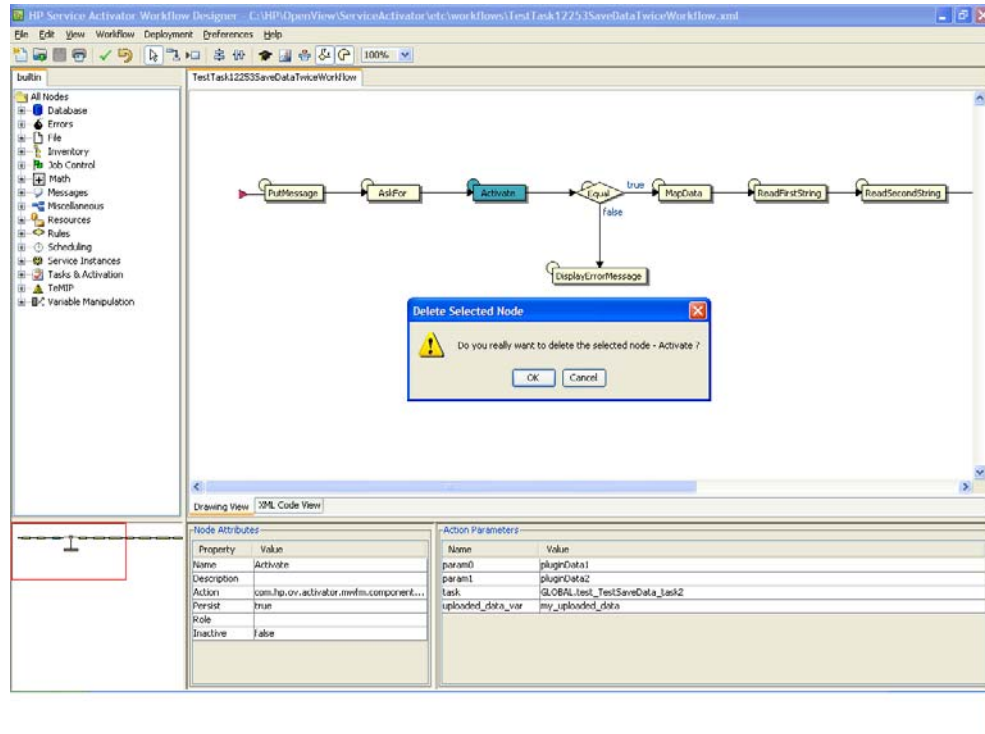


## Deleting Nodes

The Workflow Designer UI is enhanced to support single/multiple nodes deletion. Before deleting, you will be prompted with a message to confirm the deletion.

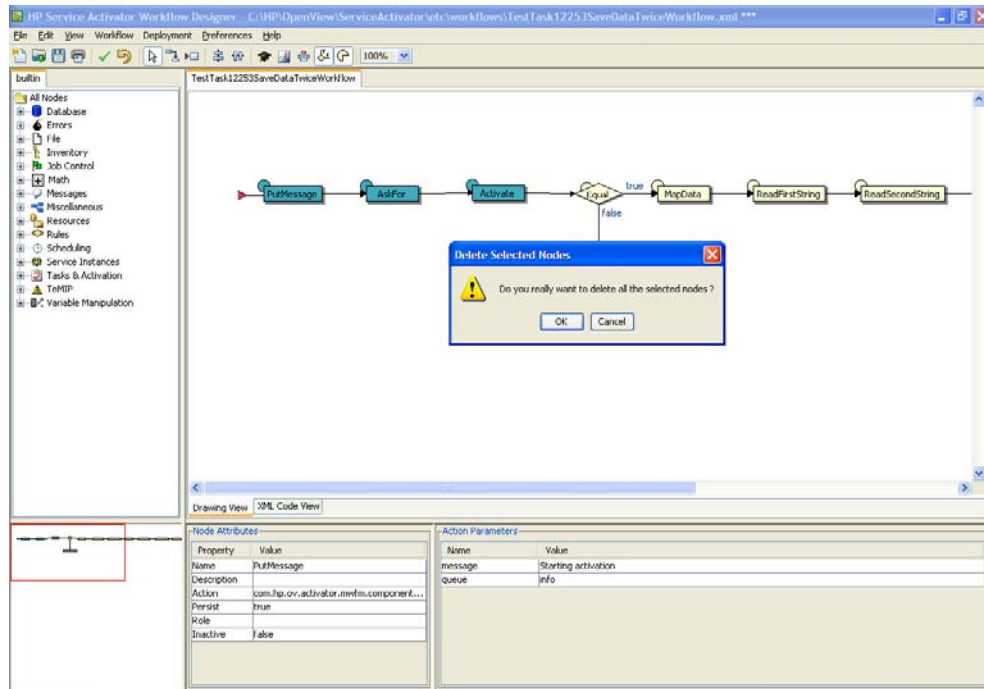
- Deleting a single node: In the Workflow Designer UI, you can select a single node and press the **Delete** key to delete the node.

**Figure 3-14** Deleting a Single Node



- Deleting multiple nodes: In the Workflow Designer UI, you can select more than one node and press the **Delete** key to delete all the selected nodes.

**Figure 3-15** Deleting Multiple Nodes

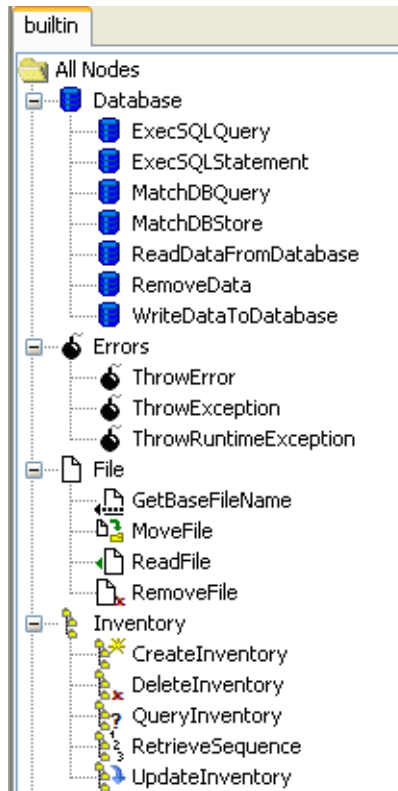


## Using the Node Tree

The node tree allows you to arrange the nodes into groups with related behavior. If needed, a node can be a member of several groups. Nodes that are not members of any group will all be listed after the last group.

The node tree view may look different on your system depending on installed solutions, your own customizations, etc.

**Figure 3-16** Workflow Node Tree



To add a new node to the workflow, select a node from this node tree and place the node in the workflow using left mouse button.

**NOTE**

Drag and drop is not supported for adding new nodes.

It is possible for the Workflow Designer to handle more than one node tree. Each subdirectory in the \$ACTIVATOR\_ETC/designer/nodes directory will translate into a new tab shown above the node tree. These tabs allow the user to easily navigate between nodes and node trees in different directories.

In each of these subdirectories the workflow nodes are grouped by their behavior (or by any other criteria decided by the user) based on the contents of the XML file workflowNodeGroups.xml which is located in the same directory as the workflow nodes.

**NOTE**

The `workflowNodeGroups.xml` file is not mandatory. If there is no such file, all nodes will simply be shown in a flat list.

By default, the `$ACTIVATOR_ETC/designer/nodes` directory contains a single subdirectory called `builtin`. The user (or solutions) can create additional directories for new nodes and group them by defining a custom `workflowNodeGroups.xml` file in each of these directories.

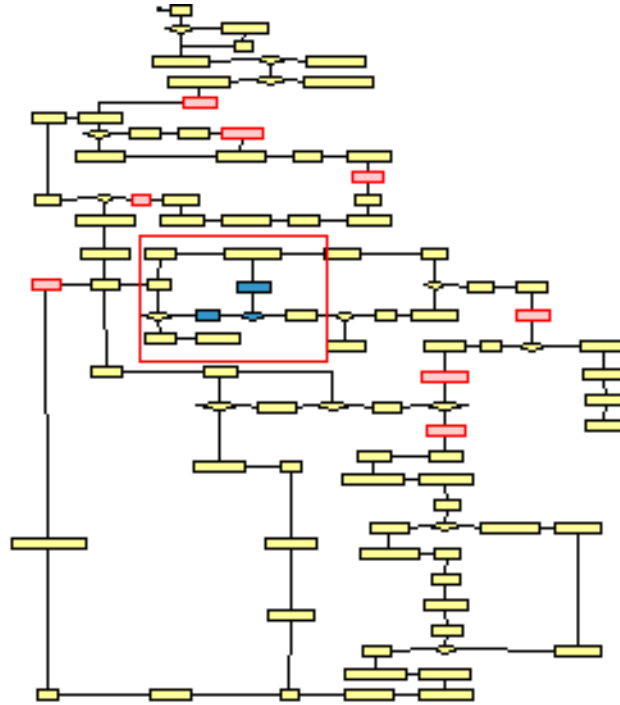
The following XML snippet shows the contents of the `workflowNodeGroups.xml` file for the two first groups shown in Figure 3-11:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE WorkflowNodeGroups SYSTEM 'workflowNodeGroups.dtd'>
<WorkflowNodeGroups>
  <Group>
    <Name>Database</Name>
    <Image>Database.gif</Image>
    <Description>Collection of Database workflow nodes</Description>
    <Nodes>
      <ClassName>com.hp.ov.activator.nwfm.component.builtin.ExecuteSQLQuery</ClassName>
      <Image>Database.gif</Image>
    </Nodes>
    <Nodes>
      <ClassName>com.hp.ov.activator.nwfm.component.builtin.ExecuteSQLStatement</ClassName>
      <Image>Database.gif</Image>
    </Nodes>
    <Nodes>
      <ClassName>com.hp.ov.activator.nwfm.component.builtin.MatchBETQuery</ClassName>
      <Image>Database.gif</Image>
    </Nodes>
    <Nodes>
      <ClassName>com.hp.ov.activator.nwfm.component.builtin.MatchBETStore</ClassName>
      <Image>Database.gif</Image>
    </Nodes>
  </Group>
  <Group>
    <Name>File</Name>
    <Image>File.gif</Image>
    <Description>Collection of File operation nodes</Description>
    <Nodes>
      <ClassName>com.hp.ov.activator.nwfm.component.builtin.GetBaseFileName</ClassName>
      <Image>Base.gif</Image>
    </Nodes>
    <Nodes>
      <ClassName>com.hp.ov.activator.nwfm.component.builtin.MoveFile</ClassName>
      <Image>MoveFile.gif</Image>
    </Nodes>
    <Nodes>
      <ClassName>com.hp.ov.activator.nwfm.component.builtin.ReadFile</ClassName>
      <Image>Read.gif</Image>
    </Nodes>
    <Nodes>
      <ClassName>com.hp.ov.activator.nwfm.component.builtin.RemoveFile</ClassName>
      <Image>RemoveFile.gif</Image>
    </Nodes>
  </Group>
  ...
</WorkflowNodeGroups>
```

---

## Using the Overview Pane

**Figure 3-17** Workflow Overview Pane



The workflow overview pane displays the entire workflow regardless of its size. At all times the overview pane shows a red box around the area of the workflow that is currently visible in the workflow view. This is particularly useful when you are editing a large workflow. Selected nodes and nodes marked as “inactive” can be identified by their color-coding; arrow heads are not displayed in the overview pane.

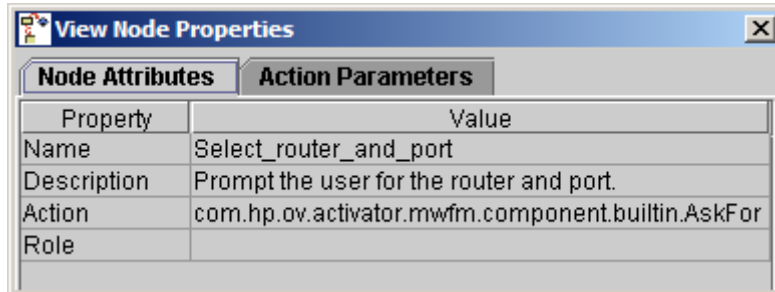
---

**NOTE** It is not possible to use the workflow overview pane to navigate the workflow.



## Using the Node Properties View

**Figure 3-18 Workflow Designer Node Properties View - Node Attributes Tab**

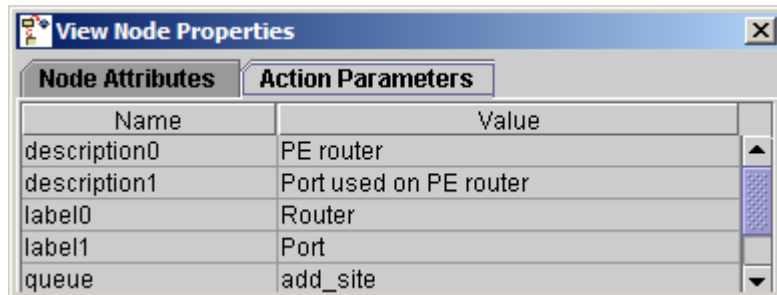


When you bring up a workflow in the design window, the node properties dialog box for that workflow is also displayed. This properties window can be used for reviewing the node properties, but also to change the properties. An other way to change the node properties is to double-click the node to bring up the Edit Node Properties dialog.

The Node Attributes tab (shown in Figure 3-18) shows the name, description, action class, and role of a node. The role is only shown for process nodes.

The Action Parameters tab (shown in Figure 3-19) shows the action parameters (name and value) for the node action. It is also here possible to edit the values directly when it is a process node.

**Figure 3-19 Workflow Designer Node Properties Window - Action Parameters Tab**

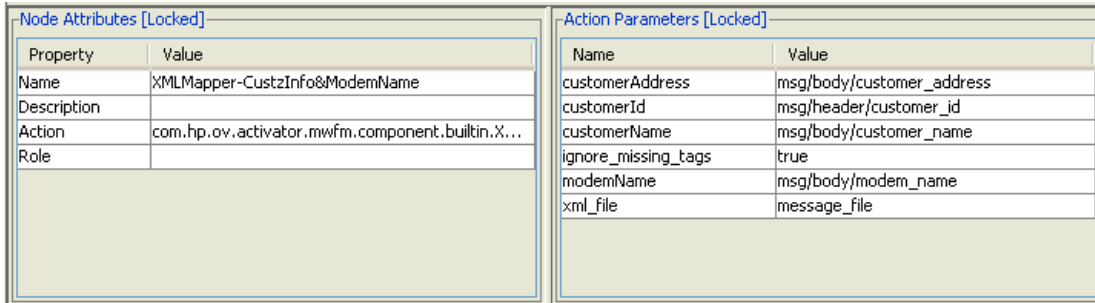


## Lock / Unlock Function

When editing a node it is sometimes convenient to be able to view the properties of another node in order to use it as a template. This is possible by using the node properties view's lock/unlock function.

The contents of the node properties view can be locked (and unlocked) by pressing the **F2** key. To indicate that the properties are locked the text “[Locked]” is appended to the border titles; an example is shown in Figure 3-15.

**Figure 3-20** Node Properties View - with Locked contents



The image shows two side-by-side panels from a software interface. The left panel is titled '-Node Attributes [Locked]' and contains a table with the following data:

Property	Value
Name	XMLMapper-CustzInfo&ModemName
Description	
Action	com.hp.ov.activator.mwfm.component.builtin.X...
Role	

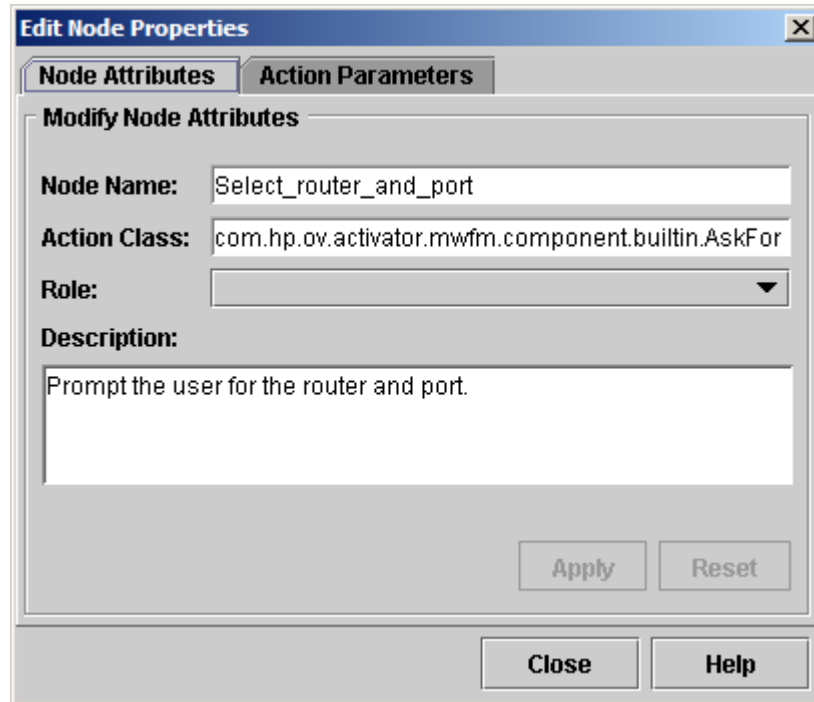
The right panel is titled '-Action Parameters [Locked]' and contains a table with the following data:

Name	Value
customerAddress	msg/body/customer_address
customerId	msg/header/customer_id
customerName	msg/body/customer_name
ignore_missing_tags	true
modemName	msg/body/modem_name
xml_file	message_file

## Using the Edit Node Properties Dialog

You may edit a workflow node's properties by either double-clicking the node you want to edit or right-clicking on that node and selecting `Edit Node Properties`.

**Figure 3-21** Workflow Designer Edit Node Properties Dialog (Node Attributes Tab)



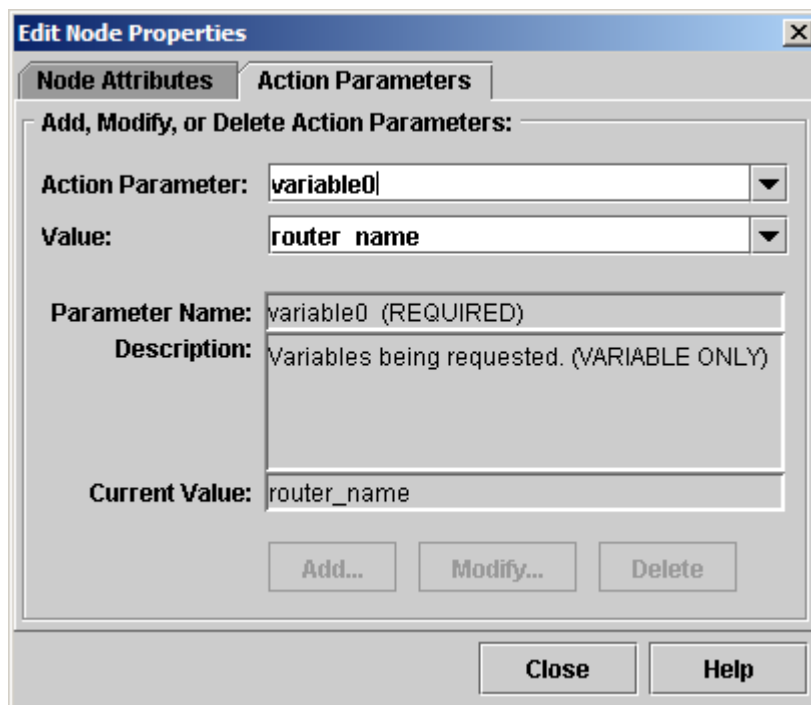
Use this tab to modify general node attributes.

- Node name  
The name is set automatically based on the node class. You may set any name you want. The name may include spaces.
- Node's action class  
Setting the action class may have unintended consequences. If the class name you specify is not recognized by the designer, then the Action Parameters tab will disappear.
- Node's assigned role (for Process Nodes)  
The role may be chosen from the current list of roles available in this workflow. To add a new role to the potential list you must edit the Workflow Settings (see page 64).
- Node's description  
The description is helpful as a documentation of the node. The description generally is not visible to an operator when the workflow is running; the exception to this is a node that performs a user interaction. When the workflow is paused waiting for input, the operator can see the description of the current node.

Make your changes and click the `Apply` button to make them permanent.  
Click the `Reset` button to revert back to the last applied state of the node.  
The `Help` button will show the definition of the current node class.

### Using the Action Parameters Tab

**Figure 3-22** Workflow Designer Edit Properties Dialog (Action Parameters Tab)



All workflow nodes are configured to perform some specific behavior by setting their parameters. A parameter consists of a name and a value. Each node has a unique set of parameters that it responds to. The node interprets those parameters appropriately to the behavior that it encapsulates. The Workflow Designer is aware of the parameters supported and required by each node.

Use the Action Parameters tab to add, modify and delete action parameters. Action parameters fall into two categories: required and optional. Required parameters must be defined for the workflow to execute properly in the workflow engine. Notice that `task` in the figure above is a required parameter, because the label `REQUIRED` is printed after the parameter name in the Parameter Name field. The value of required parameters is initialized to “You must change this.” This string will show up in the workflow’s XML code if the parameter is not given a value, and the workflow probably will not execute properly. The toolbox action “Check Workflow” will see if there are any parameters that still have a value that must be changed (see Using the Main Utilities Toolbox).

---

**NOTE**

The `Description` Field of the action parameters tab displays the generic description of the currently selected action parameter. Look there to get hints about the use and data types.

---

To modify an existing action parameter, select the `Action Parameter` text field. Either select an existing case-packet variable from the `Value` drop down menu, or type in a new value in the `Value` text field. Click the `[Modify]` button to make the change.

To add a new action parameter to the node, type the name in the `Action Parameter` text field. Then, select or type its desired value into the `Value` text field, and click `[Add]` to make the changes.

To delete an action parameter from the node, select the name in the `Action Parameter` text field. Then, click `[Delete]` to make the changes.

Remember you cannot delete `REQUIRED` action parameters. The `[Delete]` button will be disabled while a required action parameter is selected.

## Command Line Options

The Workflow Designer can be invoked from the command line using the designer script. The command line options are:

- `-version`: Display version information and exit.
- `-native`: Set native look and feel.
- `-config cfg`: Alternate configuration file.
- `-dbHost <DBHOST>`: Name of the database host. Defaults to configured db host.
- `-dbName <DBSID>`: Name of the database instance.
- `-dbPort <DBPORT>`: Database port. Default is 1521.
- `-dbUser <DBUSER>`: User name of the database instance.
- `-dbPassword <DBPASSWD>`: Password of db user name.
- `-listWorkflows`: List deployed workflows.
- `-downloadWorkflow wf`: Download the specified workflows.
- `-deleteWorkflows wf`: Mark the specified workflows as deleted.
- `-deployWorkflows wf`: Deploy the specified workflows.

---

### NOTE

On Windows; If the Workflow Designer tool is opened, the `-deployWorkflows` option cannot be used.

---

## Using Keyboard Shortcuts

Here is the list of the supported keyboard shortcuts.

<b>Shortcut</b>	<b>Purpose</b>
<b>Delete</b>	Deletes the currently selected node.
<b>CTRL-o</b>	Open Workflow—open an existing workflow
<b>CTRL-n</b>	New Workflow—create a new workflow
<b>CTRL-s</b>	Save Workflow—save the currently open workflow
<b>CTRL-p</b>	Print Workflow—print the currently open workflow
<b>CTRL-w</b>	Close Workflow—close the currently open workflow
<b>CTRL-q</b>	Switch between the open workflow views
<b>CTRL-c</b>	Copy the currently selected node (nodes) from the current workflow into the clipboard
<b>CTRL-v</b>	Paste the node (or nodes) that are currently in the clipboard into the currently active workflow
<b>CTRL-z</b>	Undo the last delete action
<b>CTRL-a</b>	Select all nodes in a workflow
<b>F2</b>	Lock/Unlock the Node Properties view
Cursor Keys (arrows)	Moves any selected nodes.





---

## **4 Workflow Node and Handler Library**

The Workflow Manager comes with an extensive library of workflow nodes and handlers that are useful to carry out many provisioning and activation tasks. Each supplied node and handler is described in detail here.

---

## Process Nodes, Rule Nodes, and Switch Nodes

This section describes the process nodes, the rule nodes, and the Switch nodes that are included in Service Activator. Each node is implemented by a Java class. The name of a node is the name of the class that implements it. Note, however, that it is a full name (including the package name) that uniquely identifies a node. All of the built-in nodes shipped with Service Activator are from the same package (`com.hp.ov.activator.mwfm.component.builtin`).

Each process node has the `throw_excep` parameter. This parameter tells the framework whether exceptions thrown inside a process node must be automatically handled or if they must be thrown, which terminates the job. Set the parameter to “true” to indicate to the system that any exception raised inside a node must terminate the job. Set the parameter to “false” to indicate to the system that all exceptions must be handled by the framework. So, if errors occur, the system will set the `RET_VALUE` case-packet variable to -1; the `RET_TEXT` case-packet variable will present an error description. The exception handling can also be controlled at the workflow level by setting the `THROW_EXCEP` system case-packet variable. This will set the behavior for all the nodes in a workflow if the `throw_excep` parameter is not set at the node level. The default value for `THROW_EXCEP` is `true`.

### Default Workflow Node Persistence Setting

All built-in workflow nodes come with a default persistence setting. The Table 4-1 shows the default persistence settings for all nodes. In the definition of the nodes, you can find the XML element `DisablePersistence` and if the value of this element is set to `FALSE`, it means that the state of the workflow job will be persisted after the node has been executed.

**Table 4-1** Default Persistence Settings for all Nodes

Node Name	Disable Persistence
Activate	FALSE
Add	TRUE
AppendToTaskList	TRUE
AppendToResourceList	TRUE
AskFor	FALSE
Assign	TRUE
Audit	FALSE
ChangeRoles	TRUE
ComposeMessage	TRUE
ConcatenateTaskLists	TRUE
ConfirmResourceReservation	TRUE
CreateInventory	FALSE

**Table 4-1 Default Persistence Settings for all Nodes**

Node Name	Disable Persistence
CreateBean	TRUE
CreateTaskList	TRUE
CreateUCMDBCIsAndRelations	FALSE
DateConverter	TRUE
DeleteCache	TRUE
DeleteInventory	FALSE
DeleteScheduleJob	FALSE
DeleteServiceInstance	FALSE
DoNothing	TRUE
DeleteUCMDBCIsAndRelations	FALSE
Decrypt	TRUE
Encrypt	TRUE
Equal	TRUE
ExecSQLQuery	TRUE
ExecSQLStatement	FALSE
ExecuteExternal	FALSE
ExecuteMacro	TRUE
ForEach	FALSE
GenericUIDialog	FALSE
GetBaseFileName	TRUE
GetBeansNNMNode	TRUE
GetBusinessHoursAfterDuration	TRUE
GetCalendarTimezone	TRUE
GetNextIncludedTime	TRUE
GetOperatingSystem	TRUE
GetTimeRangesOfBusinessDay	TRUE
GreaterThan	TRUE

**Table 4-1**                      **Default Persistence Settings for all Nodes**

<b>Node Name</b>	<b>Disable Persistence</b>
GreaterThanOrEqualTo	TRUE
HTTPGet	FALSE
HTTPRequest	FALSE
InsertIntoTaskList	TRUE
InvokeInventoryMethod	FALSE
InvokeMethod	FALSE
IsTimeIncluded	TRUE
IsTrue	TRUE
IsModule	TRUE
Java	FALSE
JavaRule	TRUE
JavaSwitch	TRUE
KillJob	FALSE
LessThan	TRUE
LessThanOrEqualTo	TRUE
Log	FALSE
MapData	TRUE
MatchDBQuery	TRUE
MatchDBStore	TRUE
MethodInvoke	FALSE
ModifyScheduledJob	FALSE
MoveFile	FALSE
MultiAssign	TRUE
Multiply	TRUE
MutexGetInfo	TRUE
MutexLock	FALSE
MutexSetInfo	FALSE
MutexUnlock	FALSE
Not	TRUE

**Table 4-1 Default Persistence Settings for all Nodes**

Node Name	Disable Persistence
NAddConfigurationPolicy	FALSE
NAddDevice	FALSE
NAddDeviceGroup	FALSE
NAddDeviceToGroup	FALSE
NAddRuleToPolicy	FALSE
NBuildConditionList	TRUE
NBuildRuleList	TRUE
NDeleteDeviceGroup	FALSE
NDeletePolicy	FALSE
NGetSnapshot	FALSE
NListConfigId	TRUE
NListDevice	TRUE
NListDeviceId	TRUE
NModifyConditionsOnRule	FALSE
NRemoveDeviceFromGroup	FALSE
NRemoveRuleFromPolicy	FALSE
NRunAdvancedScript	FALSE
NRunCommandScript	FALSE
NRunScript	FALSE
NShowConfig	TRUE
NShowDiagnostic	TRUE
NShowTask	TRUE
PatternMatch	FALSE
PAYG	FALSE
PPU	FALSE
PutMessage	FALSE
QueryInventory	TRUE
QueryScheduledJobs	TRUE
QueryServiceInstance	TRUE

**Table 4-1 Default Persistence Settings for all Nodes**

Node Name	Disable Persistence
QueryUCMDBCIsAndRelations	FALSE
QueryServiceInstanceAll	TRUE
RandomInteger	TRUE
ReadDataFromDatabase	TRUE
ReadFile	TRUE
RediscoverHostsNNMNode	TRUE
RecordOVISEvent	FALSE
ReleaseResource	FALSE
RemoveData	FALSE
RemoveFile	FALSE
Replace	TRUE
ReserveResource	FALSE
RetrieveSequence	FALSE
ScheduleCurrentJob	FALSE
ScheduleJob	FALSE
SendAlarm	FALSE
SendMessage	FALSE
SendSNMPTrap	TRUE
Sleep	TRUE
StartJob	FALSE
StartJobAndWait	FALSE
Sync	FALSE
Switch	TRUE
ThrowError	FALSE
ThrowException	FALSE
ThrowRuntimeException	FALSE
TransformXML	TRUE
UpdateBean	TRUE

**Table 4-1**      **Default Persistence Settings for all Nodes**

<b>Node Name</b>	<b>Disable Persistence</b>
UpdateCustomAttributesNNM Node	FALSE
UpdateInProgress	FALSE
UpdateInventory	FALSE
UpdateServiceInstance	FALSE
UpdateUCMDBCIsAndRelatio ns	FALSE
VariableMapper	TRUE
WasPreviousNodeOK	TRUE
WriteCasePacket	FALSE
WriteDataToDatabase	FALSE
XMLMapper	TRUE
XMLParser	TRUE

## Activate

`com.hp.ov.activator.mwfm.component.builtin.Activate`

The node contacts the activation engine to activate an atomic task, a compound task, or a task list. To invoke an atomic or compound task, you must specify the name of the task to be activated and its parameters. To invoke a previously constructed task list, you must specify the task list. See “CreateTaskList” on page 123 for additional information about using the Activate node with a task list.

The Activate node requires that the case-packet contains a variable named `activation_major_code`. After the Activate node completes, it sets the value of `activation_major_code` to indicate the status of activation. A value of 0 indicates successful activation; a value of 1 indicates an error.

If the following variables exist in the case-packet, the Activate node will also set them when activation completes:

- `activation_minor_code`
- `activation_stdout`
- `activation_stderr`
- `activation_description`

The values set for these case-packet variables are determined by aggregating the fields of the ExecutionDescriptor objects returned by all of the atomic tasks involved in the task activation.

The Activate node takes two special parameters to make it easier to respond to activation errors: `error_queue` and `error_message`. If an activation error occurs, (that is, if the `activation_major_code` is nonzero) and the `error_queue` parameter is set, then a message is posted to the specified queue. By default, the message will have the following form:

```
Error activating task "<taskname>" for job #<job_id>
```

The format of the message to be posted can be overridden by setting `error_message`. This message can be parameterized as in this example:

```
<Param name="error_message" value="Minor code %activation_minor_code% when  
activating task %activation_task%"/>
```

You can also control whether or not the Activate node really does an activation. You can control this at runtime by the values of case-packet variables. The Activate node looks in the case-packet (not a parameter) for the existence of a Boolean variable: `skip_activation`. If this has a value of “true”, the Activate node will not perform activation. In addition, if the value is “true”, then other case-packet variables are consulted to determine how to set the `activation_major_code` and `activation_minor_code`.

If `skip_activation_major_code` is set, then `activation_major_code` will be set to its value. If `skip_activation_minor_code` is set, then `activation_minor_code` will be set to its value. The `activation_description` will be set to the following string: “Activation skipped via workflow configuration. Task: + <taskname>.”



The `Activate` node can consume data uploaded from a task activation if you specify the `uploaded_data_var` parameter. See “Uploading Data from a Task Activation” on page 56 for additional information.

The `Activate` node uses the built-in variable `SUBSTEP`. It uses this to resume a workflow safely. If activation is in the middle of execution when the workflow engine is killed (not safely), the workflow engine resumes the workflow and tries to re-execute the current node. This would mean that the activation is tried again but this can be catastrophic if the activation is partially complete. To avoid this, the `Activate` node uses the `SUBSTEP` variable to record the fact that activation has actually been initiated. If the node is executed again and the `SUBSTEP` indicates this, activation is not be retried, and the node fails.

(`activation_major_code=1`, `activation_minor_code=2`...that is, `ERROR/INCONSISTENT`).

The `ActivationModule` is specially aware of the `SUBSTEP` variable. The `SUBSTEP` is not set until activation is actually begun. The `SUBSTEP` is not set until one of the activation threads actually takes the activation request from the activation queue and begins working on it.

The `Activate` node sets an entry in the `RUNTIME` variable (if it exists) to indicate the task that was executed. The key for this value is `'task_name'`.

**See Also**

- “`ActivationModule`” on page 363 in this guide
- “Job Counters” on page 96 in *HP Service Activator - Introduction and Overview*

**Table 4-2      Activate Parameters**

Name	Required	Description	Default	Type
<code>task</code>	Yes, if <code>task_list_var</code> is not used	A string indicating the name of a single atomic or compound task to execute. This parameter is mutually exclusive with the <code>task_list_var</code> parameter.	None	String
<code>param0</code> <code>param1...</code> <code>paramN</code>	Yes, if <code>task</code> is used	Specifies the values to pass for each parameter of the single task being activated. This assumes the task has been previously deployed using Service Builder. These parameters are mutually exclusive with the <code>task_list_var</code> parameter.	None	String

**Table 4-2      Activate Parameters (Continued)**

Name	Required	Description	Default	Type
<i>task_list_var</i>	Yes, if <i>task</i> is not used	<p>A case-packet variable of type Object that contains a list of tasks to execute. This object is created by the CreateTaskList node. The AppendToTaskList node is then used to add individual tasks to the task list.</p> <p>This parameter is mutually exclusive with the parameters <i>task</i> and <i>param0</i>, <i>param1</i>... <i>paramN</i>.</p>	None	Object
<i>activation_module</i>	No	Specifies the name of the activation module to use. If not specified, a module called activator is assumed.	"activator"	String
<i>error_queue</i>	No	Specifies the name of a queue to which a message will be sent if the activation fails.	None	Queue
<i>error_message</i>	No	Specifies the format of any error message that may be sent (if the <i>error_queue</i> parameter is set).	None	String
<i>ignore_lock_argument</i>	No	When set to 'true' the resource manager will not do locking on the lock arguments (Default: false)	false	Boolean
<i>uploaded_data_var</i>	No	<p>The value for this parameter must be a case-packet variable of type Object. When the Activate node is executed, this case-packet variable is set to a HashMap containing the data uploaded by the activation. If no data is uploaded during the activation, the case-packet variable is set to an empty HashMap.</p> <p>You only need to specify the <i>uploaded_data_var</i> action parameter if you wish to receive the uploaded data from the activation—it is never an error not to specify this parameter, even if the activation does, in fact, upload data.</p>	None	Object

#### Example 4-1      **Activate a Single Task**

This example invokes the UXOS\_addDir atomic task:

```
<Process-Node>
  <Name>Create a new directory</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.engine.component.builtin.Activate
    </Class-Name>
    <Param name="task" value="UXOS_addDir"/>
    <Param name="param0" value="machine"/>
    <Param name="param1" value="dirname"/>
    <Param name="param2" value="login"/>
    <Param name="param3" value="constant:users"/>
    <Param name="param4" value="constant:775"/>
    <Param name="param5" value="tarfile"/>
  </Action>
</Process-Node>
```

#### Example 4-2      **Activate a Task List**

This example activates a task list called my\_task\_list. This list was previously created using the CreateTaskList node; individual tasks were added to the list using the AppendToTaskList node.

```
<Process-Node>
  <Name>Activate the Task List Called my_task_list</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.engine.component.builtin.Activate
    </Class-Name>
    <Param name="task_list_var" value="my_task_list"/>
  </Action>
</Process-Node>
```

## Add

`com.hp.ov.activator.mwfm.component.builtin.Add`

Adds a list of values (numeric variables and constants). The operation is similar to writing a statement such as `operand0 = operand0 + operand1`. The result of the computation is stored back in the first variable.

If you specify only one variable, the node computes a simple increment, similar to writing a statement such as `operand0 = operand0 + 1`.

**Table 4-3** Add Parameters

Name	Required	Description	Default	Type
<i>op0, op1...opN</i>	Yes, at least one	Each operand (other than <i>op0</i> ), can be a case-packet variable or a constant (specified as <code>constant:X</code> where <i>X</i> is the constant). <i>op0</i> can only be a case-packet variable since that is where the result is stored.	None	Numeric

**Example 4-3** Add - use in the workflow

This example adds a list of variables and constants together. The process is similar to writing a statement such as `x = x + y + 10`.

```
<Process-Node disablePersistence="true">
  <Name>compute total time</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.engine.component.builtin.Add
    </Class-Name>
    <Param name="op0" value="x"/>
    <Param name="op1" value="y"/>
    <Param name="op2" value="constant:10"/>
  </Action>
</Process-Node>
```

## AppendToResourceList

`com.hp.ov.activator.mwfm.component.builtin.resource.AppendToResourceList`

This node adds a single resource to a list of resources. If the resource list does not exist, it will be automatically created.

The node is used in combination with the ReserveResource node to dynamically build a list of resources to be reserved in a single database transaction. I.e. the node can be used in cases where the number of resources is not known before runtime.

### See Also

- “ReserveResource” on page 281

Table 4-4

### AppendToResourceList Parameters

Name	Required	Description	Default	Type
<i>resource_list_var</i>	Yes	A case-packet variable of type Object that contains the resource list. If it is first time the node is called the Object must be null to create a new resource list.	None	Object
<i>bean</i>	Yes	Name of the Java class generated by Inventory Builder for a reservable resource from which to allocate the resource.	None	String
<i>method</i>	No	Name of the method by which the reservation is carried out. This defaults to doing a reservation using the reserveResourceByPrimarykey method	reserveResourceByPrimarykey	String
<i>field</i>	No	Name of the field by which the reservation is carried out. This defaults to doing a reservation by PrimaryKey.	Primary Key	String
<i>variable</i>	No	Name of a case-packet variable that contains the value of the key that is to be used for the reservation.	None	Depends on the bean

## AppendToTaskList

`com.hp.ov.activator.mwfm.component.builtin.tasklist.AppendToTaskList`

The node adds a single task to a list of tasks created using the `CreateTaskList` node. This task can be an atomic task or a compound task.

You can view the contents of a task list using the `PutMessage` node. This is helpful when you are debugging, as it allows you to see the contents of the task list. In the `PutMessage` node, specify a message such as “The task list is: %s” and supply the task list as `param0`.

### See Also

- “`CreateTaskList`” on page 123 for more information about creating a new task list
- “`ConcatenateTaskLists`” on page 117
- “`InsertIntoTasklist`” on page 181
- “`PutMessage`” on page 249
- “`Activate`” on page 96 for more information about the `Activate` node

**Table 4-5** **AppendToTaskList Parameters**

Name	Required	Description	Default	Type
<i>task_list_var</i>	Yes	A case-packet variable of type Object that contains the task list. Before tasks can be appended, the list must be created by <code>CreateTaskList</code> .	None	Object
<i>task</i>	Yes	Name of the atomic or compound task to add to the list. The task must have been previously deployed with Service Builder.	None	String
<i>param0</i> <i>param1...</i> <i>paramN</i>	Yes	Specifies the values to pass for each parameter of the single task being appended to the task list.	None	Depends on the task’s argument type

**Example 4-4 AppendToTaskList - use in the workflow**

This example appends a single task to a list called `my_task_list`:

```
<Process-Node disablePersistence="true">
  <Name>AppendToTaskList</Name>
  <Description>Append a Task to an Existing List</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.tasklist.AppendToTaskList
    </Class-Name>
    <Param name="task_list_var" value="my_task_list"/>
    <Param name="task" value="UXOS_addDir"/>
    <Param name="param0" value="machine"/>
    <Param name="param1" value="dirname"/>
    <Param name="param2" value="login"/>
    <Param name="param3" value="constant:users"/>
    <Param name="param4" value="constant:775"/>
    <Param name="param5" value="tarfile"/>
  </Action>
  <Next-Node>AppendToTaskList</Next-Node>
</Process-Node>
```

## AskFor

`com.hp.ov.activator.mwfm.component.builtin.AskFor`

The node causes a workflow to pause and wait for user interaction (or input from another workflow or an external process). The node places a request on a request queue and the workflow does not proceed until the request is satisfied.

You can specify a timeout period that allows the workflow to proceed without the values if the values are not submitted before the timeout period expires. If the request does timeout, the workflow sets the `TIMEOUT` variable in the case-packet to `"true"` to indicate that the timeout occurred. The workflow can then choose to take some action to deal with the timeout.

You can also specify a Java class for validating the supplied values. This Java class must implement the "Validator Interface" on page 108.

The `AskFor` node sets an entry in the `RUNTIME` variable (if it exists) to indicate the user that responded to the request. The key for this value is `'username'`.

The `AskFor` node can be configured to make the parent workflow wait on multiple children. If you want to use the functionality of the parent workflow waiting on multiple children, you have to first specify the waiting condition. This can be specified in the `wait_for_child` parameter. This parameter will take values as `"ALL"`, `"ANY"` or `"COUNT"`.

- **ALL** - This waiting condition signifies that the parent workflow is configured to wait on all the child workflows that are specified in the `"child_workflow_job_idX"` parameters.
- **ANY** - This signifies that the parent workflow is configured to wait on any one of the configured child workflows.
- **COUNT** - This signifies that the parent workflow is configured to wait on "number" of child workflows.

The `wait_for_child` parameter should be set for the parent workflow to register with the Sync module. If this is not set, the `AskFor` node will not communicate with the sync module. So, if the children spawned by the parent workflow responds to the parent workflow, the parent workflow will never know about this as the children will communicate with the sync module directly to send its responses.

If the parent workflow is configured to wait on all the child workflows, the parent workflow will look into the `"child_workflow_job_idX"` parameters and will wait in the configured queue (queue parameter) till all of them have completed before collecting the response. The response that it should send should also match the case packet that the parent workflow is waiting on. If the response from the child is to a different case packet that the parent is not expecting, the parent workflow will keep waiting even though the all the children have responded.

If the parent workflow is configured to wait on any one of the child workflows, the parent workflow will wait in the configured queue till one of the child workflow responds (child workflows are configured in the `"child_workflow_job_idX"` parameters).

If the waiting condition is `ALL` / `ANY` and the `"child_workflow_job_idX"` parameters are not set, the parent workflow will fail.

If the waiting condition is set to any other values other than `ALL` / `ANY` / `COUNT`, the parent workflow will fail.



If the parent workflow is configured to wait on a number of child workflows, the parent workflow will look into the “children\_count” parameter and will wait for the “children\_count” number of workflows to respond before collecting the response and moving out of the AskFor node. In this case, the variables “child\_workflow\_job\_idX” will not have any values. If the "children\_count" is not set and the waiting condition is COUNT, the parent workflow will fail.

**See Also**

- “Sync” on page 300 for information about how to respond to an AskFor request from another workflow.
- “SyncModule” on page 436

**Table 4-6 AskFor Parameters**

Name	Required	Description	Default	Type
<i>queue</i>	Yes	Name of the request queue in which to place the request. The value can be either a constant string or a case-packet variable.	None	String
<i>variable0,</i> <i>variable1...</i> <i>variableN</i>	Yes, at least one	One or more case-packet variables whose values are being requested.	None	String
<i>description0,</i> <i>description1 ...</i> <i>descriptionN</i>	No	You can provide a description for each requested variable. This description appears in the automatically generated form to help indicate to an operator what the value means. The value is a constant string.	None	String
<i>label0,</i> <i>label1...</i> <i>labelN</i>	No	You can provide a label for each requested variable. If you do not specify a label, the variable name is used the set the label.	None	String
<i>editable0</i> <i>editable1...</i> <i>editableN</i>	No	A Boolean value (“true” or “false”) to indicate whether the field created for this variable in the automatically generated form should be editable (“true”) or not (“false”).	“true”	Boolean
<i>required0</i> <i>required1...</i> <i>requiredN</i>	No	A Boolean value (“true” or “false”) to indicate whether a value must be supplied for each field in the automatically generated form (“true”), or if it can be left empty (“false”).	“false”	Boolean

**Table 4-6 AskFor Parameters (Continued)**

Name	Required	Description	Default	Type
<i>timeout</i>	No	The value can be a case-packet variable or a constant (specified as <code>constant:X</code> where <i>X</i> is the constant). The value indicates the amount of time (in milliseconds) to wait before proceeding to the following node in the workflow. If not specified, there is no timeout.	None	Integer
<i>response</i>	No	A constant string message that is returned once the valid values are supplied for the requested variables. The user sees this message in the Operator UI. If you set the <code>validation</code> parameter, the <code>response</code> parameter is ignored.	None	String
<i>validation</i>	No	The name of a Java class that implements the <code>Validator</code> interface. If you specify this class, the <code>validate()</code> method is invoked after the requested values are supplied. The class verifies that the values supplied are consistent. Further, if you specify this class, the <code>response</code> parameter is ignored because the <code>validate</code> method returns a response message.	None	String
<i>Wait_for_child</i>	No	Name of the case packet variable that will hold the parent waiting condition. This can have values as "ALL", "ANY" or "COUNT". This along with the combination of the below parameters are mandatory for the AskFor	None	String
<i>Children_count</i>	No	Name of the case packet variable that will hold the count of children that the parent workflow will be waiting on in case the waiting condition is "COUNT"	None	Integer

**Table 4-6 AskFor Parameters (Continued)**

Name	Required	Description	Default	Type
<i>Child_workflow_j ob_id0, Child_workflow_j ob_id1 ... Child_workflow_j ob_idN</i>	No	Case packet variables to hold the child job IDs that the parent workflow will have to wait if the waiting condition is ALL or ANY	None	
swap	No	Instructs the Workflow manager to swap-out the case-packets while the job waits in the request queue, in order to reduce memory footprint	false	Boolean

**Example 4-5 AskFor - use in the workflow**

This example waits for the operator to specify a new customer name and password.

```
<Process-Node>
  <Name>Ask for input</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.AskFor
    </Class-Name>
    <Param name="queue" value="operator_input"/>
    <Param name="variable0" value="custname"/>
    <Param name="variable1" value="passwd"/>
    <Param name="description0" value="New customer name" />
    <Param name="description1" value="User password (at least 8 chars)"/>
    <Param name="required1" value="true" />
    <Param name="response" value="New customer name and password accepted"/>
  </Action>
</Process-Node>
```

**Form Presentation**

When a user chooses to interact with a workflow waiting for input, a form is automatically generated to prompt the user for the requested values. Parameters can configure the behavior of the form indicating the following things:

- An optional description to accompany the field
- Whether each element of the form can be editable or not
- Whether the element is required

By default, the form is presented in the following way:

- Any string, numeric or object variable is presented as a text field. If the field is indicated to be not editable, then the variable is presented as static text.
- Boolean variables are presented as radio button with values true or false.
- Variables of type object get set to a string value.

## Creating Custom Forms

It is possible to override the default form that is presented. Normally, the form is presented by an internally generated JSP that is not saved. However, you can tell the system first to look for a custom JSP in the file system. If one is not found, the system will generate one on the fly and will save it to disk so that it can be edited for a custom presentation.

To enable this you must edit a parameter in the `$JBOSS_DEPLOY/hpsa.ear/activator.war/WEB-INF/web.xml` file.

1. Look for the section with the comment “Interact with running jobs”
2. Set the value of the parameter `customizeAskForNodeJSP` to “true.”
3. Optionally, set the value of the parameter `fileSavedInfo` to “true.” This will cause the generated form to present the file name in which the generated JSP is saved.

These custom JSPs must be placed in a specific location based on the name of the workflow, the step name and the queue name. The base location is indicated in the `web.xml` file. The file path is:

```
$JBOSS_DEPLOY/hpsa.ear/activator.war/customJSP/<workflow>/<stepname>/<queue>.jsp
```

---

### NOTE

If you customize one of these JSPs and then subsequently alter the node to add or remove some variables, then you will have to re-customize the page to match these changes. Also, if you change the name of the workflow, the step name or the queue, then you will have to move the customized workflow to the matching directory.

---

## Validator Interface

As indicated in the discussion of the `AskFor` node on page 104, it is possible to write a Java class that can perform some validation on a collection of case-packet variables to ensure that the appropriate values have been supplied.

The `Validator` interface has a single method that is called after a set of the requested values has been supplied. The method is passed a `HashMap` containing the set of case-packet variables and their values. The method should evaluate the set of variables for consistency and either throw a `WFInvalidCasePacketException`, or return a response message to indicate that the values are valid.

```
public interface Validator
{
    /*****
    Validates a case-packet sent by an external entity.

    @param      requestedCasePacket      The case-packet to be validated

    @return     An object holding a message to be passed up to the client

    @exception WFInvalidCasePacketException In case of any error in the returned
                                                case-packet information
    *****/
    public Object validate( HashMap requestedCasePacket ) throws
        WFInvalidCasePacketException;
}
```

## Assign

`com.hp.ov.activator.mwfm.component.builtin.Assign`

The node is a component used for assigning value to case-packet variable. Using the `VariableMapper` or the `MultiAssign` node instead is recommended because they are more flexible.

### See Also

- “`VariableMapper`” on page 323
- “`MultiAssign`” on page 216

**Table 4-7 Assign Parameters**

Name	Required	Description	Default	Type
<i>variable</i>	Yes	Case-packet variable to be set.	None	String / Integer / Float / Boolean / Object
<i>value</i>	Yes	New value to set for the variable. It can be a case-packet variable or a constant (specified as <code>constant:X</code> where <i>X</i> is the constant).	None	Depends on the variable type.

### Example 4-6

#### Assign - use in the workflow

This example sets the counter variable to a value of 0.

```
<Process-Node disablePersistence="true">
  <Name>Reset the counter</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Assign
    </Class-Name>
    <Param name="variable" name="counter" />
    <Param name="value" name="constant:0" />
  </Action>
</Process-Node>
```

## Audit

`com.hp.ov.activator.mwfm.component.builtin.Audit`

The node writes an audit record using the specified audit module.

### See Also

- “AuditModule” on page 365 for more details on how to enable or disable event types.

**Table 4-8**      **Audit Parameters**

Name	Required	Description	Default	Type
<i>audit_module</i>	No	Specifies the name of the audit module to use. If not specified, the module called “auditor” is assumed.	“auditor”	String
<i>event_type</i>	No	Specifies the type of event being audited. The type can be any string however a list of limitations exists, see the AuditModule description for details.  If not specified, “LOG_EVENT” is used. The value can be a case packet variable or a constant preceded by “constant:”	“LOG_EVENT”	String
<i>timestamp</i>	No	Specifies the time when the audited event occurred. The argument can be set to the value of a case packet variable for example the START_TIME case-packet variable or the “time” entry for a workflow step from the RUNTIME case-packet variable. If <i>timestamp</i> is not specified, the current system time is used. The format for <i>timestamp</i> is given in milliseconds. If the format of the argument is invalid, the node fails with RET_VALUE set to 1. The explanation is given in the RET_TEXT case-packet variable.	Current time	String

**Table 4-8 Audit Parameters (Continued)**

Name	Required	Description	Default	Type
<i>user</i>	No	Specifies the name of the user who is responsible for the event being audited. A case packet variable can be used, such as the “user” entry for a workflow step from the RUNTIME case packet variable. Or you can use a constant preceded by “constant:” If in your workflow you declare a case-packet variable called RUNTIME with the type Object, it will automatically be filled with some different information during the run of the workflow. The RUNTIME CP is a map. In addition to other things, it is updated with the username used at every interaction (ASKFOR) and can, therefore, be used if you want to know the user who last interacted with this job. Access to maps has been made generally available so if you need this information you can fetch it by typing RUNTIME{ “username” } in the “user” field of the audit node.	None	String
<i>step_name</i>	No	The name of the step being audited. If not specified, the name of the audit step is used. This can be a case packet variable or you can use a constant preceded with “constant:”	Audit step name	String
<i>message</i>	No	A message for the audit event. This can be a case packet variable or you can use a constant preceded by “constant:”	None	String
<i>identifier</i>	No	The Service Id for the audit event. This can be a case packet variable or you can use a constant preceded by “constant:”	None	String

**Table 4-8 Audit Parameters (Continued)**

Name	Required	Description	Default	Type
<i>attrib_name0</i> <i>attrib_name1...</i> <i>attrib_nameN</i>	No	The name of the name-value pair of data stored for this audit event. This can be either a case packet variable or a constant preceded with "constant:"  When the key is a case packet variable, the case packet variable name will be used. A lookup of the case packet variable value will not occur.  This allows expedient use of Workflow Designer to select case packet variables as names, and the use of case packet variable values as the value in the <i>attrib_value</i> parameter.	None	String
<i>attrib_value0</i> <i>attrib_value1...</i> <i>attrib_valueN</i>	No	The value of the name-value pair of data stored for this audit event. This is the name of a case packet variable or a constant preceded with "constant:"	None	Object
<i>skip</i>	No	The node will be skipped if the value is set to true.	false	Boolean

**Example 4-7 Audit - use in the workflow**

This example adds an audit record using values from the RUNTIME case packet variable for the step name "Add User".

```
<Process-Node>
  <Name>Write an Audit Record</Name>
  <Action>
    <Class-Name>
      om.hp.ov.activator.mwfm.component.builtin.Audit
    </Class-Name>
    <Param name="timestamp" value="RUNTIME{&quot;Add User&quot;}
      {&quot;timestamp&quot;}" />
    <Param name="step_name" value="constant:Add User" />
    <Param name="message" value="Add_User_Message" />
    <Param name="attrib_name0" value="constant:Task Name" />
    <Param name="attrib_value0"
      value="RUNTIME{&quot;Add User&quot;}{&quot;task_name&quot;}" />
  </Action>
</Process-Node>
```



## ChangeRoles

`com.hp.ov.activator.mwfm.component.builtin.ChangeRoles`

This node is used to change the roles dynamically within a running workflow. It is only possible to change the roles of the current job. The roles are validated against the `validroles` list set in the authentication module before changes are committed. If a role is invalid, no changes are done but the `RET_VALUE` variable is set to 1.

**Table 4-9** ChangeRoles Parameters

Name	Required	Description	Default	Type
<i>Default-Role</i>	No	The new value for the default role. This can either be a constant or a case packet variable.	None	String
<i>Trace-Role</i>	No	The new value for the trace role. This can either be a constant or a case packet variable.	None	String
<i>Kill-Role</i>	No	The new value for the kill role. This can either be a constant or a case packet variable.	None	String

**Example 4-8** ChangeRoles - use in the workflow

This example sets all tree dynamic roles of the workflow.

```
<Process-Node disablePersistence="true">
  <Name>ChangeRoles</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ChangeRoles
    </Class-Name>
    <Param name="Default-Role" value="constant:roleA"/>
    <Param name="Trace-Role" value="constant:roleB"/>
    <Param name="Kill-Role" value="constant:roleC"/>
  </Action>
</Process-Node>
```

## ComposeMessage

`com.hp.ov.activator.mwfm.component.builtin.ComposeMessage`

The node uses a template string and the current values in the case-packet to compose a new string according to the template. Placeholders in the template are replaced with the values of case-packet variables.

The template for the message can come from a file, or it can be the contents of multiple case-packet variables. If the message is to be composed from case-packet variable, multiple instances of `template_var` parameters must be mapped. The composed message can also be output to a file, or to a case-packet variable.

If the value of the `output_file` parameter indicates a file name that does not have an absolute directory path but has some directory above the file name (such as `error_messages/myfile`), the file is created under `$(ACTIVATOR_VAR)`. If the file specified is just a file name (no directory), the composed file is created under `$(ACTIVATOR_VAR)/tmp`.

The template is typically an XML message, though this is not required. The template file can contain placeholders of the form:

`%case-packet-variable-name%`

Each placeholder in the template is replaced with the value of the case-packet variable indicated in the placeholder. In the placeholder, you can also specify a default value so that it is used instead if the case-packet variable does not have a value. If the message cannot be composed for any reason, the case-packet variable `RET_VALUE` is set to 1. If the composition is successful, then `RET_VALUE` is set to 0. The syntax for the notation is:

`%case-packet-variable-name > default-value%`

**Table 4-10 ComposeMessage Parameters**

Name	Required	Description	Default	Type
<code>template_file</code>	Yes, if <code>template_var</code> is not used	Name of the file in which the template is to be found. The value of this parameter can be a case-packet variable that contains the name of the file, or can be a constant (specified as <code>constant:X</code> where <code>X</code> is the name of the file).  The file is expected to exist in the directory <code>\$(ACTIVATOR_ETC)/template_files</code>	None	String
<code>template_var0</code> , <code>template_var1</code> , ... <code>template_varN</code>	Yes, if <code>template_file</code> is not used	Name of a case-packets available that contains the template strings.	None	String

**Table 4-10 ComposeMessage Parameters (Continued)**

Name	Required	Description	Default	Type
<i>use_solution_dir</i>	No	When set to "true", the nodes will read from \$SOLUTION_ETC/template_files instead of \$ACTIVATOR_ETC/template_files.	true	Boolean
<i>output_file</i>	Yes, if <i>output_var</i> is not used	Name of the file to which the composed message is to be written. The value of this parameter can be a case-packet variable that contains the name of the file, or it can be a constant (specified as <code>constant:X</code> where <i>X</i> is the name of the file).  If the path name to the file is not an absolute path, the file is created relative to <code>\$ACTIVATOR_VAR/tmp</code>	None	String
<i>output_var</i>	Yes, if <i>output_file</i> is not used	Name of a case-packet variable in which the composed message is placed.	None	String

**Example 4-9 ComposeMessage - use in the workflow**

The following example shows the use of the ComposeMessage node.

```
<Process-Node disablePersistence="true">
  <Name>Compose success message</Name>
  <Action>
    <Class-Name>com.hp.ov.activator.mwfm.component.builtin.ComposeMessage</Class-Name>
    <Param name="output_var" value="out_message"/>
    <Param name="template_file" value="constant:OK_message.template"/>
  </Action>
  <Next-Node>Remove message file</Next-Node>
</Process-Node>
```

where the template file is located in `C:\hp\OpenView\ServiceActivator\etc\template_files`, and contains

```
<response_msg>
  <header>
    <message_id><!-- Template:message_id --></message_id>
    <service_id><!-- Template:service_id --></message_id>
  </header>

  <body>
    <message>OK</message>
  </body>
</response_msg>
```

During execution, the ComposeMessage node will substitute `<!-- Template:message_id -->` with the value of the `message_id` case-packet variable.

**Example 4-10 ComposeMessage using multiple case-packet variables**

```
<Process-Node disablePersistence= "true">
  <Name>ComposeMessage</Name>
  <Action>
    <Class-Name>com.hp.ov.activator.mwfm.component.builtin.ComposeMessage</Class-Name>
    <Param name="output_var" value="composedMessage"/>
    <Param name="template_var0" value="message_id"/>
    <Param name="template_var1" value="service_id"/>
  </Action>
  <Next-Node>PutMessage</Next-Node>
</Process-Node>
```

During execution, the ComposeMessage node will compose a string with the value of the message\_id and service\_id case-packet variables and store it in a case-packet variable composedMessage.

**Example 4-11 ComposeMessage using multiple case-packet variables, whose value contains template strings**

```
<Process-Node disablePersistence= "true">
  <Name>ComposeMessage</Name>
  <Action>
    <Class-Name>com.hp.ov.activator.mwfm.component.builtin.ComposeMessage</Class-Name>
    <Param name="output_var" value="composedMessage"/>
    <Param name="template_var0" value="message_id"/>
    <Param name="template_var1" value="servicetemplate"/>
  </Action>
  <Next-Node>PutMessage</Next-Node>
</Process-Node>
```

The case-packets messagetemplate and servicetemplate contain values message\_id and service\_id respectively.

During execution, the ComposeMessage node will substitute message\_id and service\_id with the value of message\_id and service\_id case-packet variables respectively. The composed message is stored in a case-packet variable composedMessage.

## ConcatenateTaskLists

`com.hp.ov.activator.mwfm.component.builtin.tasklist.ConcatenateTaskLists`

The node concatenates two task lists. The resulting task list is saved in the `task_list` variable, which can be activate later using the Activate node.

### See Also

- “CreateTaskList” on page 123 for more information about creating a new task list.
- “AppendToTaskList” on page 102
- “InsertIntoTasklist” on page 181
- “Activate” on page 96 for more information about the Activate node.

**Table 4-11** ConcatenateTaskLists Parameters

Name	Required	Description	Default	Type
<code>task_list_var</code>	Yes	Case-packet variable of object type holding a list of tasks to be executed.	None	Object
<code>variable</code>	Yes	Case-packet variable of object type holding a list of tasks to be appended to first list.	None	Object

**Example 4-12** ConcatenateTaskLists - use in the workflow

The following example concatenates two task lists `my_task_list` and `my_subtask_list`.

```
<Process-Node disablePersistence="true">
  <Name>ConcatenateTaskLists</Name>
  <Description>Create a Task List</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.tasklist.ConcatenateTaskLists
    </Class-Name>
    <Param name="task_list_var" value="my_task_list"/>
    <Param name="variable" value="my_subtask_list"/>
  </Action>
  <Next-Node>PutMessage</Next-Node>
</Process-Node>
```

## ConfirmResourceReservation

`com.hp.ov.activator.mwfm.component.builtin.ConfirmResourceReservation`

The node helps to manipulate the contents of the RESERVATIONS variable by removing resources from this variable. This is valuable when you choose to use the ReleaseResourceHandler to deal with abnormal workflow termination. This handler automatically releases resources in the RESERVATIONS variable.

At some point in your workflow, you might reserve a resource, then later in the workflow you actually use the resource (in an Activate node). If the workflow terminates abnormally before the Activate node is reached, you want the ReleaseResourceHandler to release the resource. However, after the Activate node runs, it is no longer appropriate to release the resource, even in an abnormal termination.

Therefore, the ConfirmResourceReservation node removes the given resource from the RESERVATIONS variable, ensuring that the ReleaseResourceHandler does not release the resource.

### See Also

- “ReserveResource” on page 281 for more information about the RESERVATIONS variable
- “ReleaseResource” on page 276 for more information about the RESERVATIONS variable

Table 4-12

### ConfirmResourceReservation Parameters

Name	Required	Description	Default	Type
<i>variable0, variable1... variableN</i>	No	Indicates the variables that contain reserved resources. If no variableN parameters are specified, all of the entries in the RESERVATIONS variable are removed.	None	Any

## CreateBean

`com.hp.ov.activator.mwfm.component.builtin.CreateBean`

This node creates and inventory bean object in memory; i.e. the object is not stored in the inventory database after being created..

**Table 4-13 CreateBean Parameters**

Name	Required	Description	Default	Type
<i>bean</i>	Yes	Name of the JavaBean class that is used for creating the inventory bean object.	None	String
<i>key_field0</i> <i>key_field1</i> ... <i>key_fieldN</i>	Yes	Name of a key in the JavaBean that is created. The parameter must be repeated for all attributes in the JavaBean being assigned. Note that when a JavaBean is updated the primary key must always be present in the list of keys.	None	String
<i>key_value0</i> <i>key_value1</i> ... <i>key_valueN</i>	Yes	Used in conjunction with the <i>key_field</i> attributes to specify the values of the individual attributes in the JavaBean.	None	Any
<i>bean_variable</i>	Yes	Name of the variable where the created JavaBean instance is returned	None	Object

**Example 4-13**      **CreateBean- use in the workflow**

This example creates in memory an inventory object representing a UNIX user.

```
<Process-Node disablePersistence="true">
  <Name>CreateUnixUser</Name>
  <Description>Create a new UNIX user</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.CreateBean
    </Class-Name>
    <Param name="bean_variable" value="user"/>
    <Param name="key_field0" value="constant:uid"/>
    <Param name="key_field1" value="constant:gid"/>
    <Param name="key_field2" value="constant:name"/>
    <Param name="key_field3" value="constant:home"/>
    <Param name="key_value0" value="next_uid_seq_number"/>
    <Param name="key_value1" value="next_gid_seq_number"/>
    <Param name="key_value2" value="user_name"/>
    <Param name="key_value3" value="home_directory"/>
  </Action>
</Process-Node>
```



## CreateInventory

`com.hp.ov.activator.mwfm.component.builtin.CreateInventory`

The node is in essence a wrapper around the `UpdateInventory` node with the only difference that it runs in “strict create” mode by default.

Values can be passed to an inventory object either by specifying a list of `key_field/key_value` pairs or by passing an object containing the inventory bean.

**Table 4-14 CreateInventory Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Name of the database module to be used.	“db”	String
<i>bean</i>	Yes	Name of the JavaBean class that is used for storing the data.	None	String
<i>bean_object</i>	No	The name of the variable containing the inventory bean object to be stored in the inventory.	None	Object
<i>key_field0,</i> <i>key_field1..</i> <i>.</i> <i>key_fieldN</i>	No	Name of a key in the JavaBean that is updated or created. The parameter must be repeated for all attributes in the JavaBean being updated or initially assigned.  Note that when a JavaBean is updated the primary key must always be present in the list of keys, even if it is not updated.	None	String
<i>key_value0,</i> <i>key_value1..</i> <i>.</i> <i>key_valueN</i>	No	Used in conjunction with the <code>key_field</code> attributes to specify the new value of the individual attributes in the JavaBean.	None	Any
<i>bean_variable</i>	No	Name of the variable where the created/updated JavaBean instance is returned.	None	Object
<i>strict_create</i>	No	When set to “true” the node will run in “strict create” mode which means that the node will fail if a bean with the specified key does already exist.  Can not be used together with the <code>strict_update</code> parameter	“true”	Boolean

**Table 4-14 CreateInventory Parameters (Continued)**

Name	Required	Description	Default	Type
<i>strict_update</i>	No	When set to “true” the node will run in “strict update” mode which means that the node will fail if a bean with the specified key does not exist.  Can not be used together with the <i>strict_create</i> parameter.	“false”	Boolean
<i>store_audit</i>	No	If audit is enabled in the Workflow Manager’s configuration file as well as in the Inventory Bean’s XML resource definition file an audit record will be written each time this node is executed.  To disable audit for the node, set this parameter to “false”.	“true”	Boolean

**Example 4-14 CreateInventory - use in the workflow**

```

<Process-Node>
  <Name>Create User</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.CreateInventory
    </Class-Name>
    <Param name="bean" value="constant:com.hp.ov.activator.triplemy.User"/>
    <Param name="db" value="db_name"/>
    <Param name="key_field0" value="constant:id"/>
    <Param name="key_value0" value="user_id"/>
    <Param name="key_field1" value="constant:firstName"/>
    <Param name="key_value1" value="user_first_name"/>
    <Param name="key_field2" value="constant:lastName"/>
    <Param name="key_value2" value="user_last_name"/>
    <Param name="key_field3" value="constant:region"/>
    <Param name="key_value3" value="region_id"/>
  </Action>
</Process-Node>
<Case-Packet>
  <Variable name="db_name" type="String"/>
  <Variable name="user_id" type="String"/>
  <Variable name="user_first_name" type="String"/>
  <Variable name="user_last_name" type="String"/>
  <Variable name="region_id" type="Integer"/>
</Case-Packet>

```

## CreateTaskList

`com.hp.ov.activator.mwfm.component.builtin.tasklist.CreateTaskList`

The node assigns a new task list to a case-packet variable of type `Object`. Tasks are then added to the list by the `AppendToTaskList` node. After the list is constructed, it is executed as a single transaction by the `Activate` node.

A task list is useful if you need to execute multiple tasks as part of a single transaction (with rollback capability), but you do not know in advance how many tasks you will need to execute. For instance, if you want to perform a certain task once for each switch in a switch fabric, but you do not know exactly how many switches there are in that fabric, you can first query your inventory to determine the number of switches and then add that number of tasks to your task list.

---

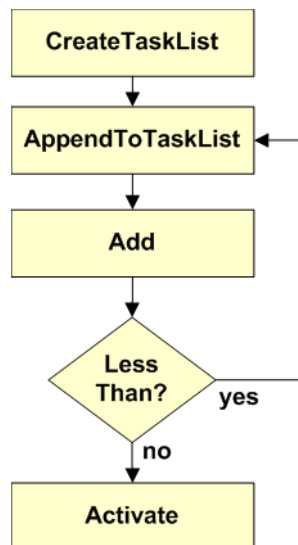
### NOTE

Task lists are not intended to replace compound tasks. You should use a compound task when you know prior to run-time which tasks you will need to execute and how many times you will need to execute each task.

---

Figure 4-1 shows a portion of a workflow that creates a task list and then appends tasks to that list. It uses a counter to keep track of the number of tasks it appends to the list, incrementing the counter once for each task. When the counter reaches a specified value, the workflow stops adding tasks to the list.

**Figure 4-1**      **Creating a Task List at Run-Time**



**See Also**

- “AppendToTaskList” on page 102 for more information about adding individual tasks to an existing task list
- “ConcatenateTaskLists” on page 117
- “InsertIntoTasklist” on page 181
- “Activate” on page 96 for more information about the Activate node

**Table 4-15 CreateTaskList Parameters**

Name	Required	Description	Default	Type
<i>task_list_var</i>	Yes	A case-packet variable of type Object that will store a task list. It is used by AppendToTaskList, ConcatenateTaskList, InsertIntoTaskList and consumed by Activate.	None	Object

**Example 4-15 CreateTaskList - use in the workflow**

The following example creates a new task list called `my_task_list`:

```
<Process-Node disablePersistence="true">
  <Name>CreateTaskList</Name>
  <Description>Create a Task List</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.tasklist.CreateTaskList
    </Class-Name>
    <Param name="task_list_var" value="my_task_list"/>
  </Action>
  <Next-Node>AppendToTaskList</Next-Node>
</Process-Node>
```

**NOTE**

If you call the `CreateTaskList` node more than once in a given workflow using the same `task_list_var` case-packet variable, any existing contents of the task list are deleted and a fresh task list is created.

## CreateUCMDBCIsAndRelations

`com.hp.ov.activator.mwfm.component.builtin.CreateUCMDBCIsAndRelations`

The createUCMDBCIsAndRelations node will create the specified CIs and Relations in the uCMDB.

This node can create multiple CIs and relations in a single request. The node sets a response variable which will be a map with the temporary id specified as the key and the actual uCMDB ID as the value. This node throws a UCMDBException in case there is an error while processing the request.

**Table 4-16 CreateUCMDBCIsAndRelations Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the UCMDBRequestModule to be used	None	String
<i>response</i>	No	The name of the case-packet variable name in which the result is stored. The type of the case-packet variable should be Object. The result returned will be a map of temporary id specified and the actual CMDB id	None	Object
ci_id0 ci_id1... ci_idN	Yes (At least one is mandatory if no relations are specified. Not mandatory if relations are specified)	Temporary Id of the CI which needs to be added. A single CI can have multiple properties. This can be specified by giving the same CI temporary Id again.	None	String
ci_type0 ci_type1... ci_typeN	Yes (If ci_id has been specified)	Type of the CI. It can be any type defined in uCMDB	None	String
ci_prop_name0 ci_prop_name1 ... ci_prop_name N	No	Name of the property to be associated with the CI	None	String

**Table 4-16 CreateUCMDBCIsAndRelations Parameters**

Name	Required	Description	Default	Type
ci_prop_value0 ci_prop_value1 ... ci_prop_value N	No	Value of the property name specified earlier. In case the property type is StringList or IntList then the property values can be a list of values. This can be specified by separating the values with the # character The ci_prop_value can also be specified as a case-packet variable. In case the property type is a StringList or an IntList then the case-packet variable has to be of type Object, Internally it can contain either a String[] or a List	None	String
ci_prop_type0	No	The type of the property. This can take the following values:  String Byte Integer Long Float Double Boolean Date XML StringList IntList	None	String
rel_id0 rel_id1... rel_idN	Yes (Atleast one is mandatory if no CIs are specified. Not mandatory if CIs are specified)	Temporary Id of the Relation which needs to be added. A single Relation can have multiple properties. This can be specified by giving the same CI temporary Id again	None	String
rel_type0 rel_type1... rel_typeN	Yes (If relation id has been specified)	Type of the Releation	None	String

**Table 4-16 CreateUCMDBCIsAndRelations Parameters**

Name	Required	Description	Default	Type
rel_end1_id1 rel_end1_id2... rel_end1_idN	Yes (If relation id has been specified)	End 1id of the relation. The ID of the CI at end 1 of the relation.	None	String
rel_end2_id1 rel_end2_id2... rel_end2_idN	Yes (If relation id has been specified)	End 2 id of the relation. The ID of the CI at end 2 of the relation.	None	String
rel_prop_name 0 rel_prop_name 1 .... rel_prop_name N	No	Name of the property to be associated with the Relation	None	String
rel_prop_value 0 rel_prop_value 1 ... rel_prop_value N	No	Value of the property name specified earlier In case the property type is StringList or IntList then the property values can be a list of values. This can be specified by separating the values with the # character. The rel_prop_value can also be specified as a case-packet variable. In case the property type is a StringList or an IntList then the case-packet variable has to be of type Object, Internally it can contain either a String[] or a List	None	String
rel_prop_type0 rel_prop_type1 ... rel_prop_type N	No	The type of the property. This can take the following values:  String Byte Integer Long Float Double Boolean Date XML StringList IntList	None	String

**Table 4-16**      **CreateUCMDBCIsAndRelations Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
date_format	No	Specifies the format in which the ci_property_value and rel_prop_values have been defined in case the property type is Date. The date format can be specified using standard java conventions used while defining a date format (as in the SimpleDateFormat class). In case this parameter is not specified then the date format is taken as the default one for the current locale in which HPSA has been deployed.	System's Locale's date format is taken	String



## DateConverter

`com.hp.ov.activator.mwfm.component.builtin.DateConverter`

This node can fetch date and time, either current time from the system, or converted from a case packet variable in any given format as string or as milliseconds since January 1, 1970 00:00:00.000 GMT.

If the time is fetched from the case packet as a user-defined date-time string it is interpreted according to the format defined in `DateStringFormat`. More information on formatting options is available below.

If time is available as a number of milliseconds since January 1, 1970 00:00:00.000 GMT then it is possible to pass this number directly to the node.

One and only one input source must be present per node. If the user does not provide any, or if several input sources are provided a configuration exception will be thrown during WF start.

It is possible to perform simple actions on the obtained data s.a. increment or decrement date and time in a flexible way. Time can be modified by a certain number of milliseconds, which can be positive or negative. Units are also allowed in the format `UNIT:NUMBER` where allowed units are: Year, Month, Day, Hour, Minute, Second. If no unit is specified milliseconds is used.

As well as for input it is possible to define the format of the desired output. Names of return parameters are self descriptive and the same rule as for input parameters exists; it is possible to have only one of them per node.

The resulting value will be saved to the workflow case packet variable specified in the result parameter after all operations on the date and time have been completed.

If a problem occurs during node execution a workflow exception will be thrown. In some workflows, where input is dynamic, it is inconvenient to break workflow execution in case of an error. To handle such issues the `throw_exception` (see beginning of section) argument should be used to control whether exceptions should be thrown or handled by setting the `RET_VALUE` and `RET_TEXT` workflow case packet variables.

The allowed format for the time format strings can be found on Sun's homepage (<http://java.sun.com>) in the API specification for the `SimpleDateFormat` class.

**Table 4-17**

### DateConverter Parameters

Name	Required	Description	Default	Type
<i>in_date_millis</i>	No	Milliseconds since January 1, 1970 00:00:00.000 GMT.  NOTE: cannot be used if <i>in_date_string</i> , <i>in_date_seconds</i> , or <i>in_current_time</i> is defined.	None	Numeric
<i>in_date_seconds</i>	No	Seconds since January 1, 1970 00:00:00.000 GMT.  NOTE: cannot be used if <i>in_date_string</i> , <i>in_date_millis</i> , or <i>in_current_time</i> is defined.	None	Numeric

**Table 4-17 DateConverter Parameters**

Name	Required	Description	Default	Type
<i>in_date_string</i>	No	Date formatted according to the <i>date_string_format</i> string.  NOTE: cannot be used if <i>in_current_time</i> or <i>in_date_millis</i> is defined. <i>date_string_format</i> must be defined.	None	Numeric
<i>in_current_time</i>	No	Initializes the node with the current time plus this amount added in milliseconds or units specified by 'Year:', 'Month:', 'Day:', 'Hour:', 'Minute:', 'Second:' (0 = now).  NOTE: cannot be used if <i>in_date_string</i> , <i>in_date_seconds</i> , or <i>in_date_millis</i> is defined.	None	String
<i>return_formatted_date</i>	No	Returns a date as string in result, formatted according to the format given here, e.g. yyyyMMddhhmm.  NOTE: cannot be used with any other Return method.	None	String
<i>return_date_field</i>	No	Returns a specific field of a date, i.e. Year, Month, MonthName, Day, DayName, Hour, Minute or Second.  NOTE: cannot be used with any other Return method.	None	String
<i>return_date_seconds</i>	No	Returns the number of seconds since January 1, 1970 00:00:00.000 GMT. The number specified here is added as seconds or units specified by 'Year:', 'Month:', 'Day:', 'Hour:', 'Minute:', 'Second:' (0 for no addition).  NOTE: cannot be used with any other Return method.	None	String
<i>return_date_millis</i>	No	Returns the number of milliseconds since January 1, 1970 00:00:00.000 GMT. The number specified here is added as milliseconds or units specified by 'Year:', 'Month:', 'Day:', 'Hour:', 'Minute:', 'Second:' (0 for no addition).  NOTE: cannot be used with any other Return method.	None	String

**Table 4-17 DateConverter Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>result</i>	Yes	String variable where returned result is to be stored.	None	String
<i>date_string_format</i>	No	The format of the in_date_string, e.g. yyyyMMddhhmm. NOTE: cannot be used without in_date_string.	None	String

**Example 4-16**      **DateConverter - use in the workflow**

This example saves the date 1970 January 1 03:01 in the form [yy-MM-dd hh:mm] to the workflow variable result.

```
<Start-Node>DateConverter</Start-Node>
<Process-Node disablePersistence="true">
  <Name>DateConverter</Name>
  <Action>

  <Class-Name>com.hp.ov.activator.mwfm.component.builtin.DateConverter</Class-Name>
  <Param name="in_date_string" value="010119700301"/>
  <Param name="date_string_format" value="MMddyyyyhhmm"/>
  <Param name="date_string_format" value="yy-MM-dd hh:mm"/>
  <Param name="result" value="result"/>

</Action>
</Process-Node>
. . .
<Case-Packet>
  <Variable name="result" type="String"/>
</Case-Packet>
```

## Decrypt

`com.hp.ov.activator.mwfm.component.builtin.Decrypt`

The node transforms an encrypted string to clear text.

Use this node just before the password must be used in clear text as it recommended to never store the password in clear text. An encrypted password can be provided e.g. from an inventory bean.

**Table 4-18**

**Decrypt Parameters**

Name	Required	Description	Default	Type
<i>encrypted_text</i>	Yes	The string to be decrypted.	None	String
<i>decrypted_text</i>	Yes	The decrypted text output (clear text)	None	String

**Example 4-17**

### Decrypt - use in the workflow

The example below uses the `Decrypt` node to transform a password to clear text.

```
<Process-Node>
  <Name>Decrypt password</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Decrypt
    </Class-Name>
    <Param name="decrypted_text" value="password"/>
    <Param name="encrypted_text" value="enc_password"/>
  </Action>
</Process-Node>

<Case-Packet>
  <Variable name="enc_password" type="String"/>
  <Variable name="password" type="String"/>
</Case-Packet>
```

## DeleteCache

`com.hp.ov.activator.mwfm.component.builtin.DeleteCache`

The node is use to delete one or all instances which are kept in the caching module. It works together with the QueryInventory node. The QueryInventory node can be configured to save the result in a caching\_module. The DeleteCache node can then be used to delete the corresponding instances again.

If the parameter delete\_all is set to true then all instances will be deleted. If delete\_all is set to false then the parameters bean, find\_by\_method, and key\_value identify which instance in the caching module should be deleted. These parameters must have the same values as when the QueryInventory node was used.

### See Also

- “QueryInventory” on page 253
- “VariableMapper” on page 323

**Table 4-19 Assign Parameters**

Name	Required	Description	Default	Type
<i> caching_mo  dule</i>	Yes	Identified which caching module the object is saved in.	None	String
<i> delete_all</i>	No	New value to set for the variable. It can be a case-packet variable or a constant (specified as <code>constant:X</code> where <i>X</i> is the constant).	false	Boolean
<i> bean</i>	Yes if delete_all is set to false	The inventory JavaBean class that was used when inserting the instance in the caching module	None	String
<i> find_by_me  thod</i>	No	Then name of the method which was used when inserting the instance in the caching module.	findByPrim aryKye	String
<i> key_value0  ,  key_value1  ...  key_valueN</i>	No	Value of the key(s) which was used when inserting the instance in the caching module.	None	Object

## DeleteInventory

`com.hp.ov.activator.mwfm.component.builtin.DeleteInventory`

The node is used to delete an instance in the inventory. It sets `RET_VALUE` to 0 if successful and to 1 if delete fails.

**Table 4-20 DeleteInventory Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Name of the database module to be used.	"db"	String
<i>bean</i>	Yes	Name of the JavaBean class that is used for deleting data.	None	String
<i>key_field0</i> , <i>key_field1...</i> <i>key_fieldN</i>	Yes	Name of a key in the JavaBean. The parameter is used to identify the data being deleted. Parameters must be repeated for each of the keys in the JavaBean.	None	String
<i>key_value0</i> , <i>key_value1...</i> <i>key_valueN</i>	Yes	Used in conjunction with the key <i>key_field</i> attributes to specify the key values of the data being deleted.	None	Any
<i>store_audit</i>	No	If audit is enabled in the Workflow Manager's configuration file as well as in the Inventory Bean's XML resource definition file, an audit record will be written each time this node is executed.  To disable audit for the node set this parameter to "false".	true	Boolean

**Example 4-18 DeleteInventory - use in the workflow**

The example below uses the `DeleteInventory` node to delete a VPN service.

```
<Process-Node>
  <Name>Delete L2 VPN</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.DeleteInventory
    </Class-Name>
    <Param name="key_field0" value="constant:ServiceId"/>
    <Param name="key_value0" value="service_id"/>
    <Param name="bean" value="com.hp.ov.activator.example.L2VPN"/>
  </Action>
</Process-Node>

<Case-Packet>
  <Variable name="service_id" type="String"/>
</Case-Packet>
```

## DeleteScheduledJob

`com.hp.ov.activator.mwfm.component.builtin.DeleteScheduledJob`

The node allows you to delete a scheduled job. You must specify the ID of the scheduled job you want to delete.

The SchedulerModule checks if the specified ID of the scheduled job exists on the list of scheduled jobs. If it exists, then the SchedulerModule deletes the job.

If the DeleteScheduledJob node finishes without errors, the RET\_VALUE case-packet variable is set to 0. Upon an error in the node, RET\_VALUE is set to 1. The RET\_TEXT case-packet variable contains more information about the problem. If you attempt to delete a job, that is not in the list of scheduled jobs, the node sets RET\_VALUE to 1, adds an error description to RET\_TEXT and continues to the next node.

### See Also

- “SchedulerModule” on page 422

**Table 4-21** DeleteScheduledJob Parameters

Name	Required	Description	Default	Type
<i>scheduled_job_id</i>	Yes	The ID of the scheduled job you want to deleted from the list of scheduled jobs.	None	Integer



## DeleteServiceInstance

`com.hp.ov.activator.mwfm.component.builtin.DeleteServiceInstance`

The node deletes service instance parameters from the service-instance repository. The unique identifier that this data is tied to is specified by means of the `service_id` parameter (name of a case-packet variable).

**Table 4-22 DeleteServiceInstance Parameters**

Name	Required	Description	Default	Type
<code>db</code>	No	Database module to use in order to perform the query.	"db"	String
<code>service_id</code>	Yes	A case-packet variable that holds the unique identifier for the service instance that is being deleted.	None	Integer

**Example 4-19 DeleteServiceInstance - use in the workflow**

The following example deletes all the service-instance parameters in the service-instance repository related to a given customer identifier (stored in the case-packet variable `customer_id`).

```
<Process-Node>
  <Name>Delete technical inventory</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.DeleteServiceInstance
    </Class-Name>
    <Param name="service_id" value="customer_id"/>
    <Param name="db" value="db"/>
  </Action>
</Process-Node>
```

## DeleteUCMDBCIAndRelations

`com.hp.ov.activator.mwfm.component.builtin.DeleteUCMDBCIAndRelations`

The deleteUCMDBCIAndRelations node will delete the specified CIs and Relations from the uCMDB.

The node throws a UCMDBException in case there is an error while processing the request.

**Table 4-23 DeleteUCMDBCIAndRelations Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the UCMDBRequestModule to be used	None	String
ci_id0 ci_id1... ci_idN	Yes (At least one is mandatory if no relations are specified. Not mandatory if relations are specified)	UCMDB Id of the CI which needs to be deleted.	None	String
ci_type0 ci_type1... ci_typeN	Yes (If ci_id has been specified)	Type of the CI. It can be any type defined in uCMDB	None	String
rel_id0 rel_id1... rel_idN	Yes (At least one is mandatory if no CIs are specified. Not mandatory if CIs are specified)	UCMDB Id of the Relation which needs to be deleted.	None	String
rel_type0 rel_type1... rel_typeN	Yes (If relation id has been specified)	Type of the Releation	None	String
rel_end1_id1 rel_end1_id2... rel_end1_idN	Yes (If relation id has been specified)	End 1id of the relation. The ID of the CI at end 1 of the relation.	None	String

**Table 4-23 DeleteUCMDBCIsAndRelations Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
rel_end2_id1 rel_end2_id2... rel_end2_idN	Yes (If relation id has been specified)	End 2 id of the relation. The ID of the CI at end 2 of the relation.	None	String

## DoNothing

`com.hp.ov.activator.mwfm.component.builtin.DoNothing`

A sample process node that simply logs a message when the node is entered and another when the node is exited.

**Table 4-24** DoNothing Parameters

Name	Required	Description	Default	Type
<i>message</i>	No	The message to be logged when the node is entered. A standard message is printed on exit.	None	String

**Example 4-20** DoNothing - use in the workflow

The following example could represent the end node of any workflow.

```
<Process-Node disablePersistence="true">
  <Name>End</Name>
  <Description>Ends workflow</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.DoNothing
    </Class-Name>
  </Action>
</Process-Node>
```

**Example 4-21** DoNothing - use in the workflow

A message is printed each time the workflow executes this node.

```
<Process-Node disablePersistence="true">
  <Name>Debug node</Name>
  <Description>Sends a message when you pass through this node</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.DoNothing
    </Class-Name>
    <Param name="message" value="*** Pass by debug node***"
  </Action>
</Process-Node>
```

## Encrypt

`com.hp.ov.activator.mwfm.component.builtin.Encrypt`

The node transforms a clear text string into an encrypted string.

This node must be used e.g. before storing an encrypted password in the inventory system.

**Table 4-25** **Encrypt Parameters**

Name	Required	Description	Default	Type
<i>text</i>	Yes	The string to be encrypted.	None	String
<i>encrypted_text</i>	Yes	The encrypted text output	None	String

**Example 4-22** **Encrypt - use in the workflow**

The example below uses the `Encrypt` node to transform a password into an encrypted string.

```
<Process-Node>
  <Name>Encrypt password</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Encrypt
    </Class-Name>
    <Param name="encrypted_text" value="enc_password"/>
    <Param name="text" value="password"/>
  </Action>
</Process-Node>

<Case-Packet>
  <Variable name="enc_password" type="String"/>
  <Variable name="password" type="String"/>
</Case-Packet>
```

## ExecuteMacro

`com.hp.ov.activator.mwfm.component.builtin.ExecuteMacro`

This node executes a workflow as a macro inside the current workflow, i.e. no new job is started. The current workflow (parent workflow) waits until the execution of macro workflow is finished.

The parameters input and output must match the workflow contract defined in the macro workflow (child workflow). The sequece must be the same as defined in the contract, but the names can be different.

Conceptually the workflow node is very similar to the `StartJobAndWait` workflow node. The difference is that no new job is started, i.e. the child workflow will be executed in the same cluster node as the parent workflow.. Also if the macro workflow node uses the swap functionalty then the case-packet variables from the parent workflow node will also be removed from memory.

In the web UI job page the workflow and step name will show the macro workflow name and the step currently executed in this workflow.

### See Also

- “Workflow Contract” on page 36

**Table 4-26** **ExecuteMacro Parameters**

Name	Required	Description	Default	Type
<i>workflow_name</i>	Yes	The name of the workflow to execute	None	String
<i>input0</i> , <i>input1</i> , ... <i>inputN</i>	Yes, if contract for macro workflow requires this	Case-packet variables that are to be passed to initialize variables in the new workflow being executed.	None	Any
<i>output0</i> , <i>output1</i> , ... <i>outputN</i>	Yes, if contract for macro workflow requires this	Case-packet variables where the output case-packet variables in the macro workflow node should be stored	None	Any

**Example 4-23** **ExecuteMacro - use in the workflow**

The example below uses the `ExecuteMacro` node to .

```
<Process-Node>
  <Name>start work</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ExecuteMacro
    </Class-Name>
    <Param name="workflow_name" value="constant:MacroChild"/>
    <Param name="input0" value="first"/>
    <Param name="input1" value="second"/>
  </Action>
</Process-Node>
```

```
    <Param name="output0" value="childOutput"/>  
  </Action>  
</Process-Node>
```

## Equal

`com.hp.ov.activator.mwfm.component.builtin.Equal`

The node allows you to compare whether variable or constant values are the same.

**Table 4-27** Equal Parameters

Name	Required	Description	Default	Type
<i>op1</i>	Yes	The two parameters are variables or constants. Constant is specified as <code>constant:X</code> . If the two variables are not of the same type, their values are converted into strings and they are compared lexically.	None	Any
<i>op2</i>	Yes	Same as above.	None	Any

**Example 4-24** Equal - use in the workflow

This example establishes whether `SendCasePacketOK` is true or false. Depending on the value, the job continues to either the `End` node or to the `Sleep` node.

```
<Rule-Node disablePersistence="true">
  <Name>Resend?</Name>
  <Description> Checks the Boolean variable SendCasePacketOK to end or resend
    the sum result
  </Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Equal
    </Class-Name>
    <Param name="op1" value="SendCasePacketOK"/>
    <Param name="op2" value="constant:true" />
  </Action>
  <True-Next-Node>End</True-Next-Node>
  <False-Next-Node>Sleep node</False-Next-Node>
</Rule-Node>
```



## ExecSQLQuery

`com.hp.ov.activator.mwfm.component.builtin.ExecSQLQuery`

This node allows a SQL query to run against a database and assigns the results to case-packet variables. The component logs warnings if the query returns no data or returns more than one row. If the query returns no data, case-packet variables are not overwritten, preserving their value. If the query returns more than one row, the extra rows are ignored.

**Table 4-28 ExecSQLQuery Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Specifies the database module to use in order to gain access to a database.	"db"	String
<i>query</i>	Yes	Query to be performed, specified as a constant string. The query can contain free variables to put data from the variables in the case-packet. A question mark is used to indicate a free variable.	None	String
<i>param0</i> , <i>param1...paramN</i>	No	If free variables have been specified in the query statement, you must supply the value of each variable. For this purpose, you can use as many <code>param</code> parameters as needed. The value of these parameters must be case-packet variable names.	None	Any
name of a case-packet variable	Yes, at least one	Once the query has been run, the values in the first row of the result can be assigned to case-packet variables. The way to specify which column goes to which variable is to indicate the name of a case-packet variable as the name of the parameter, and <code>col&lt;n&gt;</code> for the value.	None	Depends on the SQL query column type

**Example 4-25 ExecSQLQuery - use in the workflow**

This example gathers two values (`group_name` and IP address) for a web server, and stores these values in the case-packet variables named `group` and `ipaddr`, respectively.

```
<Process-Node disablePersistence="true">
  <Name>Get Web Server Details</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ExecSQLQuery
    </Class-Name>
    <Param name="query" value="select group_name, IP from
      demo_webserver where name= ?"/>
  </Action>
</Process-Node>
```

**Process Nodes, Rule Nodes, and Switch Nodes**

```
<Param name="group" value="col0"/>  
<Param name="ipaddr" value="col1"/>  
<Param name="param0" value="web-server"/>  
</Action>  
</Process-Node>
```

## ExecSQLStatement

`com.hp.ov.activator.mwfm.component.builtin.ExecSQLStatement`

The node runs an SQL statement (such as insert, update, or delete) against a database.

**Table 4-29 ExecSQLStatement Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	This parameter specifies the database module to use in order to access a database.	"db"	String
<i>statement</i>	Yes	This is the SQL statement to run. It can contain free variables to be replaced by values from case-packet variables. In this case, a question mark is used to indicate a free variable.	None	String
<i>param0</i> , <i>param1...</i> <i>paramN</i>	No	If free variables have been specified in the query statement, you must supply the value of each variable. For this purpose, you can use as many param parameters as needed.	None	Any

**Example 4-26 ExecSQLStatement**

```
<Process-Node disablePersistence="true">
  <Name>Get Web Server Details1</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ExecSQLStatement
    </Class-Name>
    <Param name="statement" value="update demo_webserver
      set port=? where server_name=?"/>
    <Param name="param0" value="port"/>
    <Param name="param1" value="web-server"/>
  </Action>
</Process-Node>
```

## ExecuteExternal

`com.hp.ov.activator.mwfm.component.builtin.ExecuteExternal`

The node runs an external program and optionally allows output from the program to be captured to set the value of case-packet variables.

In HP OVSA 4.1 version, the node was designed in a such that *param0*, *param1*, . . . *paramN* parameters was not supporting constant values. However in HP SA 5.1 version, a user can enter constants as parameters.

The command-line for the program is specified as a constant string, but it can be parameterized by replacing free variables (% s) in the statement with the value of parameters.

It is also possible to pass all or a few of the current values of the case-packet variables to the executed program on its `stdin`. By default, all of the case-packet variables are sent to the program. Alternately, you may specify a subset of the variables to be sent, using the `variableN` parameters. If you want to pass no variables to the program, set the parameter “`variable0`” to an empty string or a value of a single dash, “-”.

It is possible to capture the output from the program into a single case-packet variable with the `output_var` parameter. Additionally, if this is not specified, the output from the program is interpreted as a series of lines indicating the variable to set and its new value. The lines must be of the form:

```
variableName=newVariableValue
```

---

**NOTE**

A frequent mistake is to forget that the output from the program is treated as a list of variables and their values. Do not forget to use the `output_var` parameter if you do not want the output interpreted in this manner.

---

By default, the executed command is started with a current working directory of `$ACTIVATOR_VAR`. This can be overridden with the `cwd` parameter.

**Table 4-30**      **ExecuteExternal Parameters**

Name	Required	Description	Default	Type
<i>cmd_line</i>	Yes	The name of the program to run, along with its directory and command-line arguments. May include free variables (%s) to be replaced by <i>paramN</i> parameters.	None	String
<i>wait</i>	No	Indicates whether the node should wait for the command to complete before continuing. Specify a value of "false" if no wait is desired.	"true"	Boolean
<i>param0</i> , <i>param1</i> . . . <i>paramN</i>	No	Indicates the names of case-packet variables whose values are used to replace the free variables (%s) in the <i>cmd_line</i> . Specify as many <i>param</i> parameters as necessary ( <i>param0</i> , <i>param1</i> . . . <i>paramN</i> ).	None	Any

**Table 4-30 ExecuteExternal Parameters (Continued)**

Name	Required	Description	Default	Type
<i>output_var</i>	No	Indicates the name of a variable to capture the output of the executed program.	None	Object
<i>cwd</i>	No	Indicates the working directory in which the command should be run.	<i>\$ACTIVATOR_VAR</i>	Object
<i>variable0, variable1... variableN</i>	No	Indicates the names of case-packet variables that should be passed to the running program on its <i>stdin</i> . If unspecified, each variable in the case-packet will be sent.	None	Any

**Example 4-27 ExecuteExternal - use in the workflow**

This example copies a file.

```
<Process-Node>
  <Name>Save message file</Name>
  <Description>Copies file message_file to c:\tmp</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ExecuteExternal
    </Class-Name>
    <Param name="cmd_line" value="cmd.exe /c copy %s c:\tmp" />
    <Param name="param0" value="message_file" />
    <Param name="variable0" value="" />
  </Action>
</Process-Node>
```

## ForEach

`com.hp.ov.activator.mwfm.component.builtin.ForEach`

The node allows you to iterate over a list. The list can be an Array, Collection, or a String object. The node will iterate over the list and returns each element found in the list. Only a single element is returned at a time. The return parameter *element* gives you the iterated element. To get the next element in the list, you must recall the node. For example, if you want to get 'n' elements from the list where 'n' is the number of element, the node must be called 'n' times.

The iterated element will be stored in the return parameter *element*. The list attribute contains the original list elements and the remaining parameter contains the rest of the elements to be iterated. After the element is iterated from the list, the node will update the following parameters:

- *remaining* - contains the remaining elements in the list to be iterated.
- *idx* - contains the index of the last iterated element (index starts from 0).
- *count* - contains the number of elements iterated from the list (count starts from 1).

**Table 4-31 ForEach Parameters**

Name	Required	Description	Default	Type
<i>list</i>	Yes	The list to iterate over. A list can be an Array, Collection or a String object. The string can be a XML String, or a string where the elements are separated using the patterns defined in <code>java.util.regex.Matcher</code> , or a string where the elements are defined based on the patterns and groups defined in <code>java.util.regex.Matcher</code> .	None	String, Collection, Array Object
<i>element</i>	Yes	Contains the last iterated element from the list.		Object
<i>remaining</i>	Yes	Contains the remaining elements in the list to be iterated.	None	Object
<i>idx</i>	No	Contains the index of the last iterated element. The index starts from 0.	None	Integer
<i>count</i>	No	Contains the number of iterations. The count starts from 1.	None	Integer
<i>tag</i>	No	Contains the XML String and the elements in the string list are iterated based on the defined XML String.	None	String

**Table 4-31 ForEach Parameters (Continued)**

Name	Required	Description	Default	Type
<i>separator</i>	No	Contains the separator pattern string and the elements in the list are iterated based on the defined pattern string. Patterns are defined in <code>java.util.regex.Matcher</code> .	None	String
<i>skip_if_empty</i>	No	Used with the separator to indicate whether a completely empty string is treated as an empty list. By default, it is set to true.	"true"	Boolean
<i>pattern</i>	No	Contains the pattern string and the elements in the list are iterated based on the defined pattern string. If the pattern contains groups, only groups will be returned. Patterns and groups are defined in <code>java.util.regex.Matcher</code>	None	String

**Example 4-28 ForEach - use in the workflow**

This example illustrates an attempt by ForEach node to iterate each element in the string list, where the elements are represented by using the comma (,) separator.

```

<Rule-Node>
  <Name>ForEach</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ForEach
    </Class-Name>
    <Param name="count" value="cntVar"/>
    <Param name="element" value="elementName"/>
    <Param name="idx" value="idxVar"/>
    <Param name="list" value="listSeparatorString"/>
    <Param name="remaining" value="remainingListVar"/>
    <Param name="separator" value="separatorStr"/>
  </Action>
</Rule-Node>
...
<Case-Packet>
  <Variable name="cntVar" type="Integer"/>
  <Variable name="elementName" type="String"/>
  <Variable name="idxVar" type="Integer"/>
  <Variable name="listSeparatorString" type="String"/>
  <Variable name="remainingListVar" type="Object"/>
  <Variable name="separatorStr" type="String"/>
</Case-Packet>
...
<Initial-Case-Packet>
  <Variable-Value name="listSeparatorString" value="AB, C,"/>
  <Variable-Value name="separatorStr" value=","/>
</Initial-Case-Packet>

```

## GenericUIDialog

`com.hp.ov.activator.mwfm.component.builtin.GenericUIDialog`

This node allows the workflow manager to handle failures or display data during workflow execution.

The node can do the following:

- Present attributes in the same way as it is possible in the AskFor node
- Define options that the user can select between
- Display any kind of dialog data that can be read from a database or from a file and optionally format it too using XML style sheets
- Retrieve activation dialog data using an identifier and optionally format it using XML style sheets
- In the above options, data and identifier are mutually exclusive. Options and AskFor style attributes are not mutually exclusive. Any combination of the three options can be configured in the node

The node can be placed after any potential failure points in the workflow. For e.g. the node could be placed after an Activate node. Once an Activate node returns, check the return code and then if it a failure use a GenericUIDialog node to display the Activation dialog from the database and decide what the next course of action should be. Based on the selected choice in the UI Dialog, the user can design the workflow in so as to retry the Activate node or execute any other node. The user can not only select an option but also specify value for case-packet variables in the UI dialog.

The node causes a workflow to pause and waits for a user interaction. The node places a request on a request queue and the workflow does not proceed until the request is satisfied. If options to choose are configured in the node then the choice selected by the user is set to an output case-packet which can then be used by a Rule node, ideally a Switch node, to decide the next path

You can specify a timeout period that allows the workflow to proceed without the user selecting an option if the user does not interact before the timeout period expires. If the request does timeout, the workflow sets the `TIMEOUT` variable in the case-packet to `"true"` to indicate that the timeout occurred

The options to be displayed in the Failure Dialog when the user interacts with the job can be configured using the `option0`, `option1...optionN` parameters. Labels for these options can be specified using `option_label0`, `option_label1...option_labelN` parameters. The choice selected by the user will be set to a case-packet mapped to the parameter `output_value`.

A default output can also be specified using the parameter `default_out_value` which will also be displayed as an additional option in the UI Dialog. A default output is required if a timeout is configured and `optionN` parameters are also configured in which case this will be set as the outcome of the node.

Case-packets to be edited could be specified using the `variable0`, `variable1...variableN` parameters.



The data to be displayed can be specified using the parameter `dialog_data0`, `dialog_data1... dialog_dataN` or using an identifier. The `dialog_data0` parameter could be a plain text or URL. The URL could be a database id or a file path (an absolute path, or a filename relative to `$ACTIVATOR_VAR`) containing failure details. The syntax is `db:message_id` or `file:file_path`.

The identifier can only be a plain text.

In order to present the data the user can specify `data_tab0`, `data_tab1...data_tabN` and `xsl_url0`, `xsl_url1...xsl_urlN` parameters. The XML style sheet could be a filename relative to `$ACTIVATOR_VAR` or can be specified directly.

The `GenericUIDialog` node can be configured to swap out case-packets from memory when the job waits in a request queue by setting the `swap` parameter to `true`. This reduces the memory footprint if there are huge numbers of jobs waiting in a request queue. The list of case-packets to be retained in memory can also be specified in the `mwfm.xml`. When the user interacts with the job the swapped out case-packets are restored in the memory.

**Table 4-32 GenericUIDialog Parameters**

Name	Required	Description	Default	Type
<i>title</i>	Yes	Title of the <code>GenericUIDialog</code> Interactable window	None	String
<i>queue</i>	Yes	Queue in which the request will wait	None	String
<code>dialog_data0</code> <code>dialog_data1...</code> <code>dialog_dataN</code>	No	Stores the details about data to be displayed. It could be an exception or a message or a URL. The URL could be a database id or a file path (an absolute path, or a filename relative to <code>\$ACTIVATOR_VAR</code> ) containing failure details. The syntax is <code>db:message_id</code> or <code>file:file_path</code> . Cannot be specified if an identifier is configured	None	String
<code>identifier</code>	No	An identifier that points to activation dialogs in the database. Cannot be specified if <code>dialog_data</code> is configured	None	String
<code>dialog_label</code>	No	Label for the dialog details. Required only if <code>dialog_data</code> or <code>identifier</code> is specified	None	String

**Table 4-32 GenericUIDialog Parameters**

Name	Required	Description	Default	Type
xsl_url0 xsl_url1... xsl_urlN	No	The URL to the XML style sheet (XSL file). The URL could directly contain the style sheet or a file path (an absolute path, or a filename relative to \$ACTIVATOR_VAR) containing the style sheet. The syntax for a file name is file:file_path	None	String
data_tab0 data_tab1 data_tab2	No	The tab names	None	String
output_value	Yes, if options are specified	Stores the next course of action selected by the user	None	String
default_output_value	No	The default output that stores the next course of action if a timeout occurs before user interaction. Specify a value only if a timeout is specified and user options are also specified	None	String
option0, option1 ... optionN	No	Options that are available to the user to decide the next course of action. Necessary only if default_output_value is not specified	None	String
option_label0 option_label1... option_labelN	No	Label for the options in a GUI presentation. The number of options and labels specified must be the same	None	String
variable_label	No	Label for all the attributes being modified.	None	String
variable0, variable1 ... variableN	No	One or more case-packet variables whose values are being requested.	None	String
description0, description1 ... descriptionN	No	You can provide a description for each requested variable. This description appears in the automatically generated form to help indicate to an operator what the value means. The value is a constant string	None	String

**Table 4-32 GenericUIDialog Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
label0, label1.. labelN	No	You can provide a label for each requested variable. If you do not specify a label, the variable name is used to set the label.	None	String
editable0, editable1 ... editableN	No	A Boolean value ("true" or "false") to indicate whether the field created for this variable in the automatically generated form should be editable ("true") or not ("false").	true	Boolean
required0 required1 ... requiredN	No	A Boolean value ("true" or "false") to indicate whether a value must be supplied for each field in the automatically generated form ("true"), or if it can be left empty ("false").	false	Boolean
response	No	A constant string message that is returned once the valid values are supplied for the requested variables. The user sees this message in the Operator UI. If you set the validation parameter, the response parameter is ignored.	None	String
timeout	No	Wait time in milliseconds before jumping to the next node and setting the variable TIMEOUT=true	None	Integer
swap	No	Instructs the Workflow manager to swap-out the case-packets while the job waits in the request queue, in order to reduce memory footprint	false	Boolean

### Example 4-29 GenericUIDialog - use in the workflow

Display data by specifying dialog\_data that retrieves it from the database.

```
<Process-Node disablePersistence="true">
  <Name>GenericUIDialog</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.GenericUIDialog
    </Class-Name>
    <Param name="title" value="constant:Activation Failure dialog"/>
    <Param name="dialog_label" value="constant:Task 1 failure"/>
    <Param name="queue" value="constant:uidialog"/>
    <Param name="failure_details" value="WORKFLOW_EXCEPTION"/>
    <Param name="timeout" value="constant:20000"/>
    <Param name="swap" value="constant:true"/>
    <Param name="output_value" value="userschoice"/>
    <Param name="default_output_value" value="constant:Default"/>
    <Param name="option0" value="constant:choice1"/>
    <Param name="option1" value="constant:choice2"/>
    <Param name="option2" value="constant:choice3"/>
    <Param name="option_label0" value="constant:choose first choice"/>
    <Param name="option_label1" value="constant:choose secondchoice"/>
    <Param name="option_label2" value="constant:choose third choice"/>
    <Param name="variable_label" value="Modify Attributes..."/>
    <Param name="variable0" value="firstvar"/>
    <Param name="variable1" value="secondvar"/>
    <Param name="variable2" value="thirdvar"/>
    <Param name="dialog_data0" value="messageid1"/>
    <Param name="dialog_data1" value="messageid2"/>
    <Param name="dialog_data2" value="messageid3"/>
    <Param name="data_tab0" value="constant:tab1"/>
    <Param name="data_tab1" value="constant:tab2"/>
    <Param name="data_tab2" value="constant:tab3"/>
    <Param name="xsl_url0" value="constant:file:simplexml1_stylesheet.xsl"/>
    <Param name="xsl_url1" value="constant:file:simplexml2_stylesheet.xsl"/>
    <Param name="xsl_url2" value="constant:file:simplexml3_stylesheet.xsl"/>
  </Action>
</Process-Node>
<Case-Packet>
  <Variable name="firstvar" type="String"/>
  <Variable name="secondvar" type="String"/>
  <Variable name="thirdvar" type="String"/>
</Case-Packet>
```

### Example 4-30 GenericUIDialog - use in the workflow

Display data by specifying an identifier that retrieves multiple rows from the database.

```
<Process-Node disablePersistence="true">
  <Name>GenericUIDialog</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.GenericUIDialog
    </Class-Name>
    <Param name="title" value="constant:Activation Failure dialog"/>
    <Param name="dialog_label" value="constant:Task 1 failure"/>
    <Param name="queue" value="constant:uidialog"/>
    <Param name="failure_details" value="WORKFLOW_EXCEPTION"/>
    <Param name="timeout" value="constant:20000"/>
    <Param name="swap" value="constant:true"/>
    <Param name="output_value" value="userschoice"/>
    <Param name="default_output_value" value="constant:Default"/>
    <Param name="option0" value="constant:choice1"/>
    <Param name="option1" value="constant:choice2"/>
    <Param name="option2" value="constant:choice3"/>
    <Param name="option_label0" value="constant:choose first choice"/>
    <Param name="option_label1" value="constant:choose secondchoice"/>
    <Param name="option_label2" value="constant:choose third choice"/>
    <Param name="variable_label" value="Modify Attributes..."/>
    <Param name="variable0" value="firstvar"/>
    <Param name="variable1" value="secondvar"/>
    <Param name="variable2" value="thirdvar"/>
    <Param name="identifier" value="identifierVal"/>
    <Param name="data_tab0" value="constant:tab1"/>
    <Param name="data_tab1" value="constant:tab2"/>
    <Param name="data_tab2" value="constant:tab3"/>
    <Param name="xsl_url0" value="constant:file:simplexml1_stylesheet.xsl"/>
    <Param name="xsl_url1" value="constant:file:simplexml2_stylesheet.xsl"/>
    <Param name="xsl_url2" value="constant:file:simplexml3_stylesheet.xsl"/>
  </Action>
</Process-Node>
<Case-Packet>
  <Variable name="firstvar" type="String"/>
  <Variable name="secondvar" type="String"/>
  <Variable name="thirdvar" type="String"/>
  <Variable name="identifierVal" type="String"/>
</Case-Packet>
```

### Form Presentation

When a user chooses to interact with a workflow waiting for input, a UI Dialog form is automatically generated to prompt the user to select an option.

Parameters can configure the behaviour of the form indicating the following things:

- Labels for the user options using parameters `option_label0`, `option_label1`, `option_labelN` which will be displayed in the UI Dialog instead of the options themselves.
- A title for the UI Dialog can be specified using the parameter `title`.
- A label for the UI details can be specified using the parameter `dialog_label`
- A label for case-packets to be edited can be specified using the parameter `variable_label`

The options are displayed as radio buttons with the accompanying text set to either the option values or their corresponding labels.

The dialog data details are displayed in a non editable TextArea. .

The dialog data can also displayed in any fashion based on the style sheets and tabs configured. For e.g. if the activation dialog from a GenericCLI is to be displayed in two tabs "CLI OUTPUT" and "CLI INPUT" the values of data\_tab0 and data\_tab1 must be set to "CLI OUTPUT" and "CLI INPUT". When the first tab is clicked the activation dialog from the CLI plug-in is displayed and when the second tab is clicked the input XML sent to CLI plug-in is displayed. In order to format the data the appropriate style sheets must be specified using the parameters xsl\_url0 and xsl\_url1.

### Creating Custom Forms

It is possible to override the default form that is presented. Normally, the form is presented by an internally generated JSP that is not saved. However, you can tell the system first to look for a custom JSP in the file system. If one is not found, the system will generate one on the fly and will save it to disk so that it can be edited for a custom presentation.

To enable this you must edit a parameter in the \$JBOSS\_DEPLOY/hpsa.ear/activator.war/WEB-INF/web.xml file.

1. Look for the section with the comment "Interact with running jobs (GenericUIDialog node"
2. Set the value of the parameter customizeGenericUIDialogNodeNodeJSP to "true."
3. Optionally, set the value of the parameter fileSavedInfo to "true." This will cause the generated form to present the file name in which the generated JSP is saved.

These custom JSPs must be placed in a specific location based on the name of the workflow, the step name and the queue name. The base location is indicated in the web.xml file. The file path is:

```
$JBOSS_DEPLOY/hpsa.ear/activator.war/customJSP/<workflow>/<stepname>/<queue>
.jsp
```

## GetBaseFileName

`com.hp.ov.activator.mwfm.component.builtin.GetBaseFileName`

The node removes any path information and returns only the file name.

**Table 4-33**

### GetBaseFileName Parameters

Name	Required	Description	Default	Type
<i>file_var</i>	Yes	Variable containing the file name. The file name is placed in this variable unless <i>output_var</i> is supplied.	None	String
<i>output_var</i>	No	The optional variable which holds the file name	None	String

## GetBeansNNMNode

`com.hp.ov.activator.mwfm.component.builtin.nnmrequest.GetBeansNNMNode`

The node supports the NNM operations `getNodes`, `getInterfaces`, `getIPAddresses`, `getL2Connections` and `getIPSubnets`. As part of the call to this node from the workflow it will be necessary to specify a `beanType` node parameter in order to determine which of the five available types will be retrieved.

Depending on the `bean_type` the node will return a list of beans with the following attributes:

If a `condition_name` parameter is specified then a `condition_value` and `condition_operation` must also be specified. The same is the case for the constraint parameters. If a `constraint_name` is specified then a `constraint_value` must also be specified.

**Table 4-34**

<b>bean_type</b>	<b>Available Attributes</b>
NNM_NODE_BEAN	capabilities created customAttributes deviceCategory deviceDescription deviceFamily deviceModel deviceVendor discoveryState endNode IPv4Router id lanSwitch longName managementMode modified name notes snmpSupported snmpVersion status systemContact systemDescription systemLocation systemName systemObjectId uuid



**Table 4-34**

bean_type	Available Attributes
NNM_INTERFACE_BEAN	administrativeState capabilities connectionId created customAttributes hostedOnId id ifAlias ifDescr ifIndex ifName ifSpeed ifType managementMode modified name notes operationalState physicalAddress status uuid
NNM_L2CONNECTION_BEAN	created id modified name notes status uuid
NNM_IPSUBNET_BEAN	created id modified name notes prefix prefixLength uuid
NNM_IPADDRESS_BEAN	created hostedOnId id inInterfaceId ipSubnetId ipValue managementMode modified notes prefixLength uuid

**Table 4-35 GetBeansNNMNode Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>module_name</i>	Yes	The name of the NNMi module used to connect to a specific NNMi server	None	String
<i>bean_type</i>	Yes	The name of a bean type. The bean type can have one of the following values: NNM_NODE_BEAN NNM_INTERFACE_BEAN NNM_L2CONNECTION_BEAN NNM_IPSUBNET_BEAN NNM_IPADDRESS_BEAN	None	String
<i>result_var</i>	Yes	The case packet variable where the operation result will be stored	None	Object
<i>condition_name0,</i> <i>condition_name1,</i> .... <i>condition_nameN</i>	False	Condition name for filtering purposes	None	String
<i>condition_value0,</i> <i>condition_value1,</i> .... <i>condition_valueN</i>	False	Condition value for filtering purposes	None	String
<i>condition_operator0,</i> <i>condition_operator1,</i> .... <i>condition_operatorN</i>	False	Condition value for filtering purposes. The following values can be specified: {"EQ", "NE", "LT", "GT", "LE", "GE", "LIKE", "NOT_IN" }	None	String
<i>constraint_name0,</i> <i>constraint_name1,</i> .... <i>constraint_nameN</i>	False	The constraint name can have one of the following values (note default value is specified in parentheses): offset (0) maxObjects (1000) includeCias (false) includeCustomAttributes (false)	None	String

**Table 4-35 GetBeansNNMNode Parameters (Continued)**

Name	Required	Description	Default	Type
<i>constraint_name0,</i> <i>constraint_name1,</i> ... <i>constraint_nameN</i>	False		None	String
<i>constraint_value0,</i> <i>constraint_value1,</i> ... <i>constraint_valueN</i>	False	The parameter indicates how all the specified conditions and constrains must be joined within the expression query sent to the NNMi. This is a mandatory parameter when either conditions or constraints are specified. There are only two options for the <i>expression_operator</i> parameter: { "AND", "OR" } .	None	String

**Example 4-31 Filtering Example for GetBeansNNMNode in a workflow**

```
<Process-Node disablePersistence="true">
  <Name>GetBeansNode_example1</Name>
  <Description>
    Get all nodes (including their custom attributes). Max number of nodes retrieved to
    be 10
  </Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.nnmrequest.GetBeansNNMNode
    </Class-Name>
    <Param name="bean_type" value="constant:NNM_NODE_BEAN"/>
    <Param name="result_var" value="result"/>
    <Param name="module_name" value="constant:nnmrequest"/>
    <Param name="constraint_name0" value="constant:includeCustomAttributes"/>
    <Param name="constraint_value0" value="constant:true"/>
    <Param name="constraint_name1" value="constant:maxObjects"/>
    <Param name="constraint_value1" value="constant:10"/>
    <Param name="expression_operator" value="constant:AND"/>
  </Action>
  <Next-Node>whatever_node</Next-Node>
</Process-Node>

. . .

<Case-Packet>
  <Variable name="result" type="Object"/>
</Case-Packet>
```

**Example 4-32      GetBeansNNMNode - use in the workflow**

Get all interfaces having "name==my\_name" and "status!=NORMAL". Max interfaces retrieved to be 20.

```
<Process-Node>
  <Name>GetBeansNode_example2</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.nnmrequest.GetBeansNNMNode
    </Class-Name>
    <Param name="bean_type" value="constant:NNM_INTERFACE_BEAN"/>
    <Param name="result_var" value="result"/>
    <Param name="module_name" value="constant:nnmrequest"/>
    <Param name="condition_name0" value="constant:name"/>
    <Param name="condition_value0" value="constant:my_name"/>
    <Param name="condition_operator0" value="constant:EQ"/>
    <Param name="condition_name1" value="constant:status"/>
    <Param name="condition_value1" value="constant:NORMAL"/>
    <Param name="condition_operator1" value="constant:NE"/>
    <Param name="constraint_name0" value="constant:maxObjects"/>
    <Param name="constraint_value0" value="constant:20"/>
    <Param name="expression_operator" value="constant:AND"/>
  </Action>
  <Next-Node>whatever_node</Next-Node>
</Process-Node>
```

## GetBusinessHoursAfterDuration

`com.hp.ov.activator.mwfm.component.builtin.businesscalendar.GetBusinessHoursAfterDuration`

This node calculates the business time after a particular number of hours or minutes using the Business Calendar Module.

The result returned is a string or an integer depending on the case-packet variable type.

In case this variable is of type Integer, then the value returned will be the date, time value represented as milliseconds since January 1, 1970.

In case the variable type is string then the value returned is a date string. In case the `date_format` parameter has been specified, the result will be formatted with the same date format. Otherwise the output string will have the default date format of the locale of the system in which Service Activator is running.

Each calendar has a defined time-zone. In case the input time is in a different time-zone, then the conversion to the calendar's time-zone will be taken care of by the node, if the `timezone` parameter has been specified. The value of this parameter can be any of the values which are defined by the java `TimeZone` API (the values returned by the `TimeZone.getAvailableIds` method in the `java.util` package).

Table 4-36

### GetBusinessHoursAfterDuration Parameters

Name	Required	Description	Default	Type
<i>response</i>	Yes	The name of the case-packet variable name in which the result is stored. The type of the case-packet variable should be either of type Integer or String. The actual business time after the given duration is populated in the case-packet variable.	None	String/ Integer
<i>calendar_name</i>	Yes	The name of the calendar which needs to be used.	None	String
<i>date_value</i>	Yes	The start time to which the duration specified needs to be added. This parameter must have the date and the time specified as a string (including AM/PM, if applicable). The string can either be in the format of the default locale or conform to the date format specified by the <code>date_format</code> parameter.	None	String

**Table 4-36 GetBusinessHoursAfterDuration Parameters (Continued)**

Name	Required	Description	Default	Type
<i>duration</i>	Yes	This parameter indicates the duration which will be added to the value of the <i>date_value</i> parameter to get the business hour. The value of this parameter must be a positive integer.	None	Integer
<i>time_unit</i>	No	The allowed values are "hours" or "minutes".	minutes	String
<i>date_format</i>	No	Specifies the format in which the <i>date_value</i> parameter has been defined. The date format can be specified using standard java conventions for defining a date format (as in the SimpleDateFormat class). In case this parameter is not specified the current locale is used.	System' Locale's date format is used	String
<i>timezone</i>	No	The timezone in which the <i>date_value</i> is specified. If this parameter is specified then the time is converted to the calendar's timezone. The value of this parameter must be any of the values used by the java TimeZone class.	Current timezone of the system in which Service Activator is running.	String

**Example 4-33 GetBusinessHoursAfterDuration - In a workflow**

To get business hours 1 hour from Aug 21, 2009 1:30 pm (specified in the default format of US locale), which is defined as working day in a calendar named "calendarName". The business hours are from 8 am to 6 pm.

With no date format specified:

```

<Process-Node disablePersistence="true">
  <Name>GetBusinessHoursAfterDuration</Name>
  <Description></Description>
  <Action>
    <Class-Name>

com.hp.ov.activator.mwfm.component.builtin.businesscalendar.GetBusinessHoursAfterD
uration
    </Class-Name>
    <Param name="calender_name" value="constant:calendarName"/>
    <Param name="date_value" value="constant:Aug 21, 2009 1:30:00 PM"/>
    <Param name="duration" value="constant:1"/>
    <Param name="response" value="response_var"/>
    <Param name="time_unit" value="constant:hours"/>
  </Action>
</Process-Node>
    
```

The value returned will be Aug 21, 2009 2:30:00 PM.

```
<Process-Node disablePersistence="true">
  <Name>GetBusinessHoursAfterDuration</Name>
  <Description></Description>
  <Action>
    <Class-Name>

com.hp.ov.activator.mwfm.component.builtin.businesscalendar.GetBusinessHoursAfterD
uration
    </Class-Name>
    <Param name="calender_name" value="constant:calendarName"/>
    <Param name="date_format" value="constant:ddMMyyyyhhmmaa"/>
    <Param name="date_value" value="constant:210820090130pm"/>
    <Param name="duration" value="constant:1"/>
    <Param name="response" value="response_var"/>
    <Param name="time_unit" value="constant:hours"/>
  </Action>
</Process-Node>
```

## GetCalendarTimezone

`com.hp.ov.activator.mwfm.component.builtin.businesscalendar.GetCalendarTimezone`

The node lets the user find out which timezone the calendar is set to.

**Table 4-37** GetCalendarTimezone Parameters

Name	Required	Description	Default	Type
<i>response</i>	Yes	The name of the case-packet variable name in which the result is stored. The type of the case-packet variable should be String.	None	String/ Integer
<i>calendar_name</i>	Yes	The name of the calendar which needs to be used.	None	String

**Example 4-34** GetCalendarTime - In a workflow

To get the calendar time zone.

```
<Process-Node disablePersistence="true">
  <Name>GetCalendarTimezone</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.businesscalendar.GetCalendarTimezone
    </Class-Name>
    <Param name="calender_name" value="constant:calendarName"/>
    <Param name="response" value="response_var"/>
  </Action>
</Process-Node>
```



## GetNextIncludedTime

`com.hp.ov.activator.mwfm.component.builtin.businesscalendar.GetNextIncludedTime`

The node calculates the next business time using the Business Calendar Module.

The node takes a given time and calculates the start time for the next business hour. In case the time specified is within the business hours, then the value returned is the same time. In case the value given is after the business hours or falls on a holiday, then the next time which is in the business hours is returned.

The result returned is a string or an integer depending on the case-packet variable type.

In case this variable is of type Integer, then the value returned will be the date, time value represented as milliseconds since January 1, 1970.

In case the variable type is string then the value returned is a date string. In case the `date_format` parameter has been specified, then the result will be formatted with the same date format. Otherwise the output string will have the default date format of the locale of the system in which Service Activator is running.

Each calendar has a defined time-zone. In case the input time is in a different time-zone, then the conversion to the calendar's time-zone will be taken care of by the node, if the `timezone` parameter has been specified. The value of this parameter can be any of the values which are defined by the java `TimeZone` API (the values returned by the `TimeZone.getAvailableIds` method in the `java.util` package).

**Table 4-38** GetNextIncludedTime Parameters

Name	Required	Description	Default	Type
<i>response</i>	Yes	The name of the case-packet variable name in which the result is stored. The type of the case-packet variable should be either of type Integer or String. The next included time is populated in the case-packet variable.	None	String/ Integer
<i>calendar_name</i>	Yes	The name of the calendar which needs to be used.	None	String
<i>date_value</i>	Yes	The time after which the next included time needs to be found.  This parameter must have the date and the time specified as a string (including AM/PM, if applicable). The string can either be in the format of the default locale or conform to the date format specified by the <code>date_format</code> parameter	None	String

**Table 4-38**      **GetNextIncludedTime Parameters (Continued)**

Name	Required	Description	Default	Type
<i>date_format</i>	No	Specifies the format in which the <i>date_value</i> parameter has been defined. The date format can be specified using standard java conventions for defining a date format (as in the SimpleDateFormat class). In case this parameter is not specified the current locale is used.	System' Locale's date format is used	String
<i>timezone</i>	No	The timezone in which the <i>date_value</i> is specified. If this parameter is specified then the time is converted to the calendar's timezone. The value of this parameter must be any of the values used by the java TimeZone class.	Current timezone of the system in which Service Activator is running.	String

**Example 4-35**      **GetNextIncludedTime - In a workflow**

To get the next business time after Aug 17, 2009 5 pm, which is defined as working day in a calendar named "calendarName". The business hours are from 8 am to 6 pm.

```

<Process-Node disablePersistence="true">
  <Name>GetNextIncludedTime</Name>
  <Description></Description>
  <Action>
    <Class-Name>

com.hp.ov.activator.mwfm.component.builtin.businesscalendar.GetNextIncludedTime
    </Class-Name>
    <Param name="calender_name" value="constant:calendarName"/>
    <Param name="date_value" value="constant:170820090500pm"/>
    <Param name="date_format" value="ddMMyyyyhhmmaa"/>
    <Param name="response" value="response_var"/>
  </Action>
</Process-Node>
    
```

The value returned will be 170820090500pm.

## GetTimeRangesOfBusinessDay

```
com.hp.ov.activator.mwfm.component.builtin.businesscalendar.GetTimeRangesOfBusinessDay
```

The node retrieves the start and end times of a given day using the Business Calendar Module.

The node takes a given time and calculates start and end times for the day of the week as defined in the business calendar. The start time is populated in the case-packet variable specified using the `start_time_range` parameter and the end time is populated in the case-packet variable specified using the `end_time_range` parameter. It is not mandatory to specify both of these parameters. In case only one of these parameters have been specified, only the corresponding value is populated.

The result returned is a string or an integer depending on the case-packet variable types. If the `start_time_range` and the `end_time_range` variables are of type Integer and the `date_value` has been specified then the value returned the start/end time on the date specified represented as milliseconds since January 1, 1970.

If the variable types are of type string then the value returned is a date string. In case the `date_format` parameter has been specified, then the result will be formatted with the same date format. Otherwise the output string will have the default date format (with only the time) of the locale of the system in which Service Activator is running.

In case both the `day_of_week` and `date_value` parameter have been specified then the value specified by `date_value` parameter is taken into account and the other is ignored.

**Table 4-39** GetTimeRangesOfBusinessDay Parameters

Name	Required	Description	Default	Type
<code>start_time_range</code>	No	The name of the case-packet variable in which the start time of the day is stored. The type of the case-packet variable should be either of type Integer or String.  In case <code>day_of_week</code> parameter has been specified then the casepacket variable type must be string.	None	String/ Integer
<code>end_time_range</code>	No	The name of the case-packet variable in which the end time of the day is stored. The type of the case-packet variable should be either of type Integer or String.  In case <code>day_of_week</code> parameter has been specified then the casepacket variable type must be string.	None	String/ Integer

**Table 4-39 GetTimeRangesOfBusinessDay Parameters (Continued)**

Name	Required	Description	Default	Type
<i>calendar_name</i>	Yes	The name of the calendar which needs to be used.	None	String
<i>date_value</i>	Yes	This parameter specifies the day of the week on which the start and end time ranges need to be found. The valid values for this parameter are "sunday", "monday", "tuesday", "wednesday", "thursday", "friday", "saturday"	None	String
<i>date_format</i>	No	Specifies the format in which the date_value parameter has been defined. The date format can be specified using standard java conventions for defining a date format (as in the SimpleDateFormat class). In case this parameter is not specified the current locale is used.	System's Locale's date format is taken	String

**Example 4-36 GetTimeRangesOfBusinessDay - In a workflow**

To get the calendar time zone.

```

<Process-Node disablePersistence="true">
  <Name>GetTimeRangesOfBusinessDay</Name>
  <Description></Description>
  <Action>
    <Class-Name>

com.hp.ov.activator.mwfm.component.builtin.businesscalendar.GetTimeRangesOfBusinessDay

    </Class-Name>
    <Param name="calendar_name" value="constant:calendarName"/>
    <Param name="date_format" value="constant:ddMMyyyyhhmmaa"/>
    <Param name="date_value" value="constant:070920090500pm"/>
    <Param name="end_time_range" value="end_time"/>
    <Param name="start_time_range" value="start_time"/>
  </Action>
</Process-Node>
    
```

The value returned will be 070920090800am and 070920090600pm.

## GetOperatingSystem

`com.hp.ov.activator.mwfm.component.builtin.GetOperatingSystem`

The node allows the mwfm to provide a means to retrieve the operating system on which the current workflow is running.

The operating system type is retrieved and stored in a case packet that is mapped to the action parameter “output\_var.”

**Table 4-40** **GetOperatingSystem Parameters**

Name	Required	Description	Default	Type
<i>output_var</i>	Yes	The name of the case packet variable to return the operating system type.	None	String
<i>throw_except</i>	No	Controls whether the node should throw exceptions upon failures, or the framework should handle them. If set to 'false' the framework handles the failure by setting the RET_VALUE case packet variable to -1. The RET_TEXT variable will hold the failure text. (Default is 'true')	None	String

**Example 4-37** **GetOperatingSystem**

This example retrieves the operating system.

```
<Process-Node disablePersistence="true">
  <Name>GetOperatingSystem</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.GetOperatingSystem
    </Class-Name>
    <Param name="output_var" value="operatingSystem" />
  </Action>
</Process-Node>
```

## GreaterThan

`com.hp.ov.activator.mwfm.component.builtin.GreaterThan`

The node allows you to establish whether a variable or a constant is strictly greater than another. It works with all types of variables. If two variables are of different types, they are compared like strings.

**Table 4-41**      **GreaterThan Parameters**

Name	Required	Description	Default	Type
<i>op1</i>	Yes	The two parameters are variables or constants. Constant is specified as <code>constant:X</code> . If the two variables are not of the same type, their values are converted into strings and they are compared lexically.	None	Numeric
<i>op2</i>	Yes	Same as above.	None	Numeric

**Example 4-38**      **GreaterThan - use in the workflow**

This example determines whether the value of *var1* is strictly greater than 0.

```
<Rule-Node disablePersistence="true">
  <Name>Greater than?</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.GreaterThan
    </Class-Name>
    <Param name="op1" value="var1"/>
    <Param name="op2" value="constant:0"/>
  </Action>
  <True-Next-Node>Greater than</True-Next-Node>
  <False-Next-Node>Less or equal</False-Next-Node>
</Rule-Node>

. . .

<Case-Packet>
  <Variable name="var1" type="Integer"/>
</Case-Packet>
```

## GreaterThanEqual

`com.hp.ov.activator.mwfm.component.builtin.GreaterThanEqual`

The node allows you to establish whether a variable or a constant is greater than or equal to another. It works with all types of variables. If two variables are of different types, they are compared like strings.

**Table 4-42** **GreaterThanEqual Parameters**

Name	Required	Description	Default	Type
<i>op1</i>	Yes	The two parameters are variables or constants. Constant is specified as <code>constant:X</code> . If the two variables are not of the same type, their values are converted into strings and they are compared lexically.	None	Numeric
<i>op2</i>	Yes	Same as above.	None	Numeric

**Example 4-39** **GreaterThanEqual - use in the workflow**

The following example determines whether the value of `var1` is greater than or equal to 0.

```
<Rule-Node disablePersistence="true">
  <Name>Greater than or equal?</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.GreaterThanEqual
    </Class-Name>
    <Param name="op1" value="var1"/>
    <Param name="op2" value="constant:0"/>
  </Action>
  <True-Next-Node>Greater than or equal to</True-Next-Node>
  <False-Next-Node>Strictly less</False-Next-Node>
</Rule-Node>

. . .

<Case-Packet>
  <Variable name="var1" type="Integer"/>
</Case-Packet>
```

## HTTPGet

`com.hp.ov.activator.mwfm.component.builtin.HTTPGet`

The node is used to send a HTTP(S) GET request to some target server and receives a response. It also supports the following additional features.

- HTTPS Server/Client side certificates
- Proxy server
- HTTP basic username/password authentication for network connection
- Customizable timeout value
- Cookies

---

### NOTE

This node must be used with care. It must only be used when the time to perform the operation is very limited. The reason for this is because the workfer thread which is used to execute the node is NOT freed when sending the HTTP get request.

---

**Table 4-43 HTTPGet Parameters**

Name	Required	Description	Default	Type
<i>URL</i>	Yes	The target URL for the HTTP(S) connection.	None	String
<i>username</i>	No	Username for network connection authentication	None	String
<i>password</i>	No	Password for network connection authentication	None	String
<i>keystore</i>	No	The location of the keystore file, necessary for HTTPS client authentication.	None	String
<i>storepass</i>	No	The password to access the keystore file, necessary for HTTPS client authentication.	None	String
<i>keypass</i>	No	The password for the public certificate/private key pair, necessary for HTTPS client authentication.  Note that the parameters <i>keystore</i> , <i>storepass</i> , and <i>keypass</i> must all be set to some non-empty values for the plug-in to do HTTPS client authentication; otherwise there will be no effect.	None	String
<i>proxy_server</i>	No	Name of a proxy server.	None	String



**Table 4-43 HTTPGet Parameters (Continued)**

Name	Required	Description	Default	Type
<i>proxy_port</i>	No	Port of a proxy server.  Note that the parameters <i>proxy_server</i> and <i>proxy_port</i> must both be set to some non-empty values in order for the plug-in to set up the proxy connection. Setting only one of them will not have any effect.	None	String
<i>cookie</i>	No	Cookie of the HTTP(S) request	None	String
<i>connect_timeout</i>	No	Connection timeout value, in milliseconds.	None	String
<i>read_timeout</i>	No	Read timeout value, in milliseconds.	None	String
<i>response</i>	Yes	Case-packet variable holding the HTTP(S) response.	None	String
<i>return_cookie</i>	No	Case-packet variable holding the returned cookie, if any.	None	String

**Example 4-40 HTTPGet - use in the workflow**

This example uses the HTTPGet node to send HTTP(S) request to some target server and receive response. The result is saved in the case packet variable *response*, and the returned cookie is saved in the case packet variable *return\_cookie*, if any..

```

<Process-Node>
  <Name>Send HTTP GET Request</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.HTTPGet
    </Class-Name>
    <Param name="URL" value="targetURL"/>
    <Param name="username" value="username"/>
    <Param name="password" value="password"/>
    <Param name="keystore" value="keystore"/>
    <Param name="storepass" value="storepass"/>
    <Param name="keypass" value="keypass"/>
    <Param name="proxy_server" value="proxyhost"/>
    <Param name="proxy_port" value="proxyport"/>
    <Param name="cookie" value="cookie"/>
    <Param name="connect_timeout" value="connectTimeout"/>
    <Param name="read_timeout" value="readTimeout"/>
    <Param name="response" value="response"/>
    <Param name="return_cookie" value="cookie_result"/>
  </Action>
</Process-Node>

<Case-Packet>
  <Variable name="targetURL" type="String"/>
  <Variable name="username" type="String"/>
  <Variable name="password" type="String"/>

```

```
<Variable name="keystore"    type="String"/>
<Variable name="storepass"  type="String"/>
<Variable name="keypass"    type="String"/>
<Variable name="proxyhost"  type="String"/>
<Variable name="proxyport"  type="String"/>
<Variable name="cookie"     type="String"/>
<Variable name="connectTimeout" type="String"/>
<Variable name="readTimeout"  type="String"/>
<Variable name="response"     type="String"/>
<Variable name="cookie_result" type="String"/>
</Case-Packet>
```

## HTTPRequest

`com.hp.ov.activator.mwfm.component.builtin.HTTPRequest`

The node is used to send a HTTP(S) GET or POST request to some target server and receives a response.

The following parameters need to set to make a HTTP(s) request:

- `request_type`: The type of request. It can either POST or GET
- `response`: Returned result for the HTTP(S) GET or POST request

The node makes the HTTP(S) request using a `HTTPSenderModule`.

The message to be sent to the HTTP server in case of a POST can be specified using the parameter `request`. This can either be a message or a file URL of a file containing the message. For the second case, the URL must start with "file://". This must be specified in case of a POST.

Cookie returned by the HTTP server can be stored in a case-packet by specifying the parameter `return_cookie`. The returned cookie can be used by subsequent GET or POST request to enable the HTTP server to track the request.

The `HTTPSenderModule` process the request asynchronously thus freeing up the worker thread. The module posts the job in a request queue and sends the response once the HTTP(S) is processed.

In case of successful processing the response sent by the HTTP server is set to the case-packet mapped to `response` and any returned cookie is set to the corresponding case-packet.

In case of a failure the `RET_VALUE` is set to 1 to indicate failure and the exception is logged.

**Table 4-44 HTTPRequest Parameters**

Name	Required	Description	Default	Type
<code>request_type</code>	Yes	The type of request. It can either POST or GET	None	String
<code>module</code>	Yes	The name of the HTTP module	None	String
<code>request</code>	No	The message to be sent to the HTTP server. This can either be a message or a file URL of a file containing the message. For the second case, the URL must start with "file://". This must be specified in case of a POST	None	String
<code>return_cookie</code>	No	Returned cookie value	None	String
<code>response</code>	Yes	Returned result for the HTTP(S) GET or POST request	None	String

**Example 4-41      HTTPRequest - use in the workflow**

```
<Process-Node>
  <Name>HTTP Request</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.HTTPRequest
    </Class-Name>
    <Param name="module" value="constant:http_example_sender"/>
    <Param name="request_type" value="constant:GET"/>
    <Param name="response" value="response"/>
    <Param name="return_cookie" value="cookie_result"/>
  </Action>
</Process-Node>
```

**Example 4-42      HTTPRequest - use in the workflow**

```
<Process-Node>
  <Name>HTTP Request</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.HTTPRequest
    </Class-Name>
    <Param name="module" value="constant:http_example_sender"/>
    <Param name="request_type" value="constant:POST"/>
    <Param name="response" value="postResult"/>
    <Param name="request" value="constant:HP Service Activator"/>
  </Action>
</Process-Node>
```

## InsertIntoTasklist

`com.hp.ov.activator.mwfm.component.builtin.tasklist.InsertIntoTasklist`

The node is used to insert a task into a task list at a specified position. The task can then be activated using the `Activate` node.

### See Also

- “CreateTaskList” on page 123 for more information about creating a new task list.
- “ConcatenateTaskLists” on page 117
- “Activate” on page 96 for more information about the `Activate` node.
- “AppendToTaskList” on page 102 for more information about appending a task to the end of a task list.

**Table 4-45** InsertIntoTasklist Parameters

Name	Required	Description	Default	Type
<i>task_list_var</i>	Yes	Indicates the variable containing the task list to insert into. (Created using <code>CreateTaskList</code> )	None	Object
<i>task</i>	Yes	A variable or a constant containing the name of the task to be inserted in the list.	None	String
<i>param0</i> , <i>param1</i> , ..., <i>paramN</i>	Yes	Specifies the values of the parameters for the task being inserted. At least one value must be specified.	None	Depends on the task parameter type
<i>position</i>	No	Position of the new task in the list after insert is performed.	0	Integer

**Example 4-43** InsertIntoTasklist - use in the workflow

The following example inserts the task `my_task` in the top of the task list `my_subtask_list`.

```
<Process-Node disablePersistence="true">
  <Name>InsertToTasklist</Name>
  <Description>Create a Task List</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.tasklist.InsertIntoTasklist
    </Class-Name>
    <Param name="task_list_var" value="my_task_list"/>
    <Param name="position" value="constant:0"/>
    <Param name="task" value="constant:my_task"/>
    <Param name="param0" value="constant:my_task_param0"/>
  </Action>
</Process-Node>
```

**Process Nodes, Rule Nodes, and Switch Nodes**

```
    </Action>  
    <Next-Node>CreateTaskList</Next-Node>  
</Process-Node>
```

## InvokeInventoryMethod

`com.hp.ov.activator.mwfm.component.builtin.InvokeInventoryMethod`

The node used to invoke an arbitrary method on the inventory bean object. This node relies on the JavaBeans generated by the InventoryBuilder tool. The first argument of the method must always be a database connection object. It sets the `RET_VALUE` to 0 if the method returns an object different from `null`; otherwise the `RET_VALUE` is set to 1. No explicit commit is done.

**Table 4-46 InvokeInventoryMethod Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Name of the database module to be used.	"db"	String
<i>bean</i>	Yes	Name of the JavaBean to be used.	None	String
<i>arg0</i> <i>arg1...</i> <i>argN</i>	Yes	Arguments to be passed to the method. If its value begins with <code>constant</code> : the key is the value provided after this. Otherwise, the key indicates the name of a variable that holds the key value.	None	Depend on the bean
<i>variable</i>	Yes	Case-packet variable holding the invoked method's return value.	None	String
<i>method</i>	No	Name of the method to be invoked.	"findByPrimaryKey"	String

## InvokeMethod

`com.hp.ov.activator.mwfm.component.builtin.InvokeMethod`

The node used to invoke an arbitrary method on an object. The object may be a JavaBean generated by the InventoryBuilder tool or it may be any other Java object. The method may be static or dynamic.

The method is allowed (but not required) to take one database connection parameter in any position, as long as the type is `java.sql.Connection`. If such a parameter exists in the method, the `InvokeMethod` node will automatically supply a database connection as specified by the "db" parameter given to the node.

By default, the connection object will not be in autocommit-mode, but when the method execution completes, the node will always issue a `commit()` call on the connection. This means that the method may use `commit()` and `rollback()` methods on the connection as needed. If the method throws an exception, the node will automatically call `rollback()` instead of `commit()`.

The node automatically converts the `arg0..argN` parameters to the expected parameter types of the method following the same rules as the `JavaNode`. Since most parameter conversions are possible, overloading of Java methods should be avoided, except on the number of paramters.

If the method returns a value (i.e. is not void), the return value may be captured in an optional "variable" parameter. The node automatically converts the return value to the type of the given case-packet variable. String representations of numbers can be converted to int, long, etc. Conversion to boolean type supports the following values (not case-sensitive):

**Table 4-47**

Type	Value	boolean	Comment
String	"true"	true	
String	"yes"	true	
String	"enabled"	true	
String	"0"	true	
String	"false"	false	
String	"no"	false	
String	"disabled"	false	
String	"1"	false	
String	""	false	Only return value - not supported for arguments



**Table 4-47**

Type	Value	boolean	Comment
String	“ ”	false	Only return value; any number of whitespace characters
String	“ ”	false	Only return value; any number of whitespace characters
int	0	true	Same for Integer, long, float, etc.
int	1	true	Same for Integer, long, float, etc.
int	null	true	
int	not null	true	

Note that 0 is interpreted as true in order to comply with the handling of RET\_VALUE. In other contexts, this may be a surprising conversion.

If the method throws an exception, the node follows the conventions of the standard `throw_excep` parameter.

**Table 4-48 InvokeMethod Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Name of the database module to be used if the method takes a Connection argument.	“db”	String
<i>bean</i>	Yes	A case-packet variable containing the object on which to invoke the method. Alternatively, its value can begin with constant: followed by the fully qualified name of the class on which to invoke a static method	None	String/ Object

**Table 4-48 InvokeMethod Parameters (Continued)**

Name	Required	Description	Default	Type
<i>arg0</i> <i>arg1...</i> <i>argN</i>	No	Arguments to be passed to the method. If its value begins with constant: the key is value provided after this. Otherwise, the key indicates the name of a variable that holds the key value. If the method takes a parameter of type <code>java.sql.Connection</code> , it is passed implicitly, and should not be listed among the <code>arg</code> parameters to <code>InvokeMethod</code>	None	Any
<i>variable</i>	Yes	Case-packet variable holding the invoked method's return value.	None	Any
method	Yes	Name of the method to be invoked.	None	String

## IsModuleConfigured

`com.hp.ov.activator.mwfm.component.builtin.IsModuleConfigured`

The node checks if a specified Workflow Manager module is configured.

**Table 4-49**

### IsTrue Parameters

Name	Required	Description	Default	Type
<i>module</i>	Yes	The name of the module to be checked.	None	String

## IsTimeIncluded

`com.hp.ov.activator.mwfm.component.builtin.businesscalendar.IsTimeIncluded`

The node returns true or false depending on whether or not the provided time is within business hours.

Each calendar has a defined time-zone. In case the input time is in a different time-zone, then the conversion to the calendar's time-zone will be taken care of by the node, if the `timezone` parameter has been specified. The value of this parameter can be any of the values which are defined by the java `TimeZone` API (the values returned by the `TimeZone.getAvailableIDs` method in the `java.util` package).

**Table 4-50** IsTimeIncluded Parameters

Name	Required	Description	Default	Type
<i>response</i>	Yes	The name of the case-packet variable name in which the result is stored.	None	String/ Integer
<i>calendar_name</i>	Yes	The name of the calendar which needs to be used.	None	String
<i>date_value</i>	Yes	The time after which the next included time needs to be found.  This parameter must have the date and the time specified as a string (including AM/PM, if applicable). The string can either be in the format of the default locale or conform to the date format specified by the <code>date_format</code> parameter.	None	String
<i>date_format</i>	No	Specifies the format in which the <code>date_value</code> parameter has been defined. The date format can be specified using standard java conventions for defining a date format (as in the <code>SimpleDateFormat</code> class). In case this parameter is not specified the current locale is used.	System' Locale's date format is used	String
<i>timezone</i>	No	The timezone in which the <code>date_value</code> is specified. If this parameter is specified then the time is converted to the calendar's timezone. The value of this parameter must be any of the values used by the java <code>TimeZone</code> class.	Current timezone of the system in which Service Activator is running.	String

**Example 4-44**      **IsTimeIncluded - In a workflow**

To test if Aug 17, 2009 5 pm is within business hours, which is a defined as working day in a calendar named "calendarName". The business hours are from 8 am to 6 pm.

```
<Process-Node disablePersistence="true">
  <Name>IsTimeIncluded</Name>
  <Description></Description>
  <Action>
    <Class-Name>

com.hp.ov.activator.mwfm.component.builtin.businesscalendar.IsTimeIncluded
    </Class-Name>
    <Param name="calender_name" value="constant:calendarName"/>
    <Param name="date_value" value="constant:170820090500pm"/>
    <Param name="date_format" value="ddMMyyyyhhmmaa"/>
    <Param name="response" value="response_var"/>
  </Action>
</Process-Node>
```

The value returned will be true.

## IsTrue

`com.hp.ov.activator.mwfm.component.builtin.IsTrue`

The node checks if a case-packet variable of type Boolean is true or false.

**Table 4-51**      **IsTrue Parameters**

Name	Required	Description	Default	Type
<i>op1</i>	Yes	The name of a Boolean case-packet variable, whose value must be tested.	None	Boolean

## Java

`com.hp.ov.activator.mwfm.component.builtin.JavaNode`

This is a process node designed to execute the Java code contained in a template file, or embedded as a Java string, or a simple Java expression. The result can be saved in a case-packet variable. The node parameters can be found in Table 4-52.

The `javacode` or `javafile` parameter is used to generate a body of the class in which the desired method is declared. The `javacode` parameter is intended for only a minor code block as it is difficult to use the workflow designer for writing large blocks of Java code. The `javafile` parameter can be conveniently used to generate complex classes with a number of methods. At the same time, the content of the `javacode` parameter adheres to the same rules as the content of the `javafile` parameter.

The `javafile` parameter references a java template file. It works similarly to other HP Service Activator templates (e.g. XSL): it is read at every node call; if the template changes, the changes take effect immediately. Just like in case of XSL, the filename can be taken directly from a case-packet variable. This allows parameterization of the template name.

If necessary, it is possible to pass arguments to the created method from the workflow. The number of the `arg0-argN` must fit during runtime – type conversion is tried as good as possible. Overloaded methods, therefore, should be used with caution. Type conversion is also used for the `output_var`.

The class name is generated as a summary of the content of the class body. It is only recompiled if the content changes and loaded only if it has not already been loaded to avoid time penalty. The used compilation string is “`javac -d <dirname> -classpath classpath <filename>`”. The classpath is formed during the startup of HP Service Activator.

If scope rules are disabled (`strict_scope` is set to false), all case-packet variables available within the workflow are created as member variables in the generated class and are initialized with current values. Therefore, all case-packet variables are available as member-variables in the generated class. They can be type safely reached directly from the methods of the class (or from the expression parameter).

If scope rules are enabled, all builtin HP Service Activator variables can be seen and changed (unless declared constant), but variables originating from the workflow cannot be seen or changed unless the variable is listed in the `in_scope` list.

Changed values are automatically transferred back to the case-packets of the workflow.

As mentioned above, new `.java` and `.class` files are created every time the code is changed. If the called method throws either an in-compile or run-time exception, the process node will throw a `WFException` and the workflow execution will terminate abnormally. This of course depends on the `THROW_EXCEP` settings for the workflow or the node. If it is setup to handle exception automatically, then the `RET_VALUE` will be set to -1 and the `RET_TEXT` will contain the exception text.

The generated files are placed in the path specified by `Dyn-Class-Path` in the `mwfm.xml` configuration file. If the path is not specified, it defaults to `$ACTIVATOR_VAR/Dyn`. It is impossible to detect when dynamic generated files become outdated. They are left on the disc and must be deleted if necessary. They will reappear if needed. Deleting such files too often causes performance problems due to compilation time.

The format of the previously mentioned template file differs from Java because it has to reserve room for the auto generated name of the class and code.

The template format is described below. Schematically it can be presented as follows:

```

<jtp> ::= <import>* [<extends>] <implements>* <class-body>
<import> ::= "import" <package> [";"] "\n"
<extends> ::= "extends" <class-name> [";"] "\n"
<implements> ::= "implements" <interfaces> [";"] "\n"
<interfaces> ::= <interface-name> ["," <interfaces>]
<class-body> ::= any valid java code
    
```

Comments are allowed only in the class body part. The file must always start with the above described keywords. If anything else than the expected keyword is found, it is assumed to be java declarations. Examples of the java template file and the generated code can be found in Example 4-46 on page 194 and Example 4-47 on page 194.

**Table 4-52**      **Java Parameters**

Name	Required	Description	Default	Type
<i>javafile</i>	No	Name on the java template file passed in the string case-packet variable or as a constant. The default place to look for the template is \$ACTIVATOR_ETC/template_files. The default file extension is .jtp (java template). It can be combined with method name as <file>::<method> in order to force specified method to be called for execution. But it must not be combined with the method parameter.	None	String
<i>javacode</i>	No	The contents of a java template as defined by <jtp> format. Note that it should only be used for minor operations.	None	String
<i>method</i>	No	Name of the method to call, which can be case-packet variable of constant. If it is omitted, the method name is derived from javafile or expression.	None	String
<i>expression</i>	No	Simple Java expression.	None	String



**Table 4-52 Java Parameters (Continued)**

Name	Required	Description	Default	Type
<i>arg0...argN</i>	No	Parameters or arguments to pass to the method. Care should be taken to pass the correct number of parameters taken by the method. This parameter cannot be combined with expression.	None	Depends on the method argument type
<i>output_var</i>	No	Optional variable for the return value. Type of the variable should coincide with return type of the executed method.	None	Object
<i>cleanup</i>	No	Optional string containing the name of a method to be executed in the NodeExited method. See javadoc Class WFProcessNode.	None	String
<i>strict_scope</i>	No	This Java node adheres to scoping rules(As explained above)	True	Boolean
<i>in_scope</i>	No	List of variables brought into scope if strict_scope is true.	None	String
<i>use_solution_dir</i>	No	When set to "true", the nodes will read from \$SOLUTION_ETC/template_files instead of \$ACTIVATOR_ETC/template_files.	false	Boolean

**Example 4-45 Java - use in the workflow**

This example compiles and executes Java code generated according to the data found in the template file \$ACTIVATOR\_ETC/template\_files/Attributes.jtp. The method chosen for execution is sms(). It takes an argument of type String. The argument is set in arg0 as constant string "any". The output is saved in the out\_var case-packet variable.

```
<Process-Node>
  <Name>Java</Name>
  <Action>
    <Class-Name>com.hp.ov.activator.mwfm.component.builtin.JavaNode</Class-
    <Param name="output_var" value="out_var"/>
    <Param name="javafile" value="filename"/>
    <Param name="arg0" value="any"/>
  </Action>
</Process-Node>
```

```
<Case-Packet>
  <Variable name="filename" type="String"/>
</Case-Packet>
<Initial-Case-Packet>
  <Variable-Value name="filename" value="Attributes.jtp:sms"/>
</Initial-Case-Packet>
```

#### Example 4-46 Java Template (jtp) File

```
import com.fut.byf.*
extends Goo;
implements Foo, Bar;
implements Baz

public void f() {
  x = x + 1;
}
```

---

#### NOTE

The example is not intended to provide a valid code. It rather shows where the template declarations can be found in the generated file. Therefore, compilation of the Java file generated according to the template will fail. The case-packet variable `x` should be defined in the workflow in order for this code to work. Assuming that the case-packet variable is of type `Integer`, the generated code will be similar to the code shown in Example 4-47.

---

#### Example 4-47 Generated Java Code

```
package com.hp.ov.activator.dyn;

// DYNAMICALLY GENERATED CLASS - please do not edit!!!
import com.hp.ov.activator.mwfm.component.*;
import com.hp.ov.activator.mwfm.component.builtin.*;
import com.hp.ov.activator.mwfm.component.builtin.java.*;
import com.hp.ov.activator.mwfm.engine.object.*;
import com.hp.ov.activator.mwfm.engine.module.*;
import java.sql.*;
import java.util.*;
import com.fut.byf.*;

public class Dyn2529779203829335138 extends Goo implements DynNodeIF, Foo, Bar, Baz
{
  public Dyn2529779203829335138()
  {}

  WFContext wfContext;
  private long x = 0;
  public void f() {
    x = x + 1;
  }
}
```

```
}
```

Much of the code for handling initialization and reversing case-packet variables follows the code in Example 4-47. Please note that direct access to the `WFContext` is available as a member variable. There are some utility methods provided to enable easy access to the database.

```
private DatabaseModule getDBModule(String db) { ... }  
private Connection getConnection(String db) { ... }  
private Connection getConnection() { return getConnection("db"); }
```

The connection is released automatically in a `cleanup_` method in the auto-generated code, which makes it impossible for the database connection to remain unreleased.

## JavaRule

`com.hp.ov.activator.mwfm.component.builtin.JavaRule`

The JavaRule node works in the same way as the Java node. The main difference is that JavaRule is a rule node, which processes Boolean expressions. Therefore, it is convenient for use in condition branching. The node parameters are presented in Table 4-53.

The node returns WFEException and workflow execution fails if:

- A runtime exception is returned during code execution although the code has been compiled and run successfully.
- The result of code execution cannot be converted to a Boolean.

The true or false branch of the node adheres to the following rules:

- Boolean or boolean, directly from the value.
- String understands “true”, “false”, “yes”, “no”, “enabled”, “disabled” (case insensitive).
- Object is converted to String and hereafter handled as such.
- Integer will give true if its value is 0 (zero).
- If the value cannot be converted according the rules above, method execution will throw a WFEException, and workflow execution will fail

### See Also

- “Java” on page 191 for more information about the parameters and Java code usage

**Table 4-53**      **JavaRule Parameters**

Name	Required	Description	Default	Type
<i>javafile</i>	No	Name of the java template file. If used as <file>::<method>, then the method specified is called instead of the one specified by the method parameter.	None	String
<i>javacode</i>	No	Java functional declaration.	None	String
<i>method</i>	No	Name of the method to call.	None	String
<i>expression</i>	No	Java expression. Expression can be combined with javafile/javacode and can be wrapped in an extra dummy method.	None	String

**Table 4-53 JavaRule Parameters (Continued)**

Name	Required	Description	Default	Type
<i>arg0...argN</i>	No	Parameters or arguments to pass to the method; cannot be combined with expression.	None	Depends on the method argument type
<i>strict_scope</i>	No	This Java node adheres to scoping rules(As explained above)	True	Boolean
<i>in_scope</i>		List of variables brought into scope if strict_scope is true.	None	String
<i>use_solution_dir</i>	No	When set to "true", the nodes will read from \$SOLUTION_ETC/template_files instead of \$ACTIVATOR_ETC/template_files.	false	Boolean

**Example 4-48 JavaRule - use in the workflow**

```
<Rule-Node disablePersistence="true">
  <Name>JavaRule</Name>
  <Action>
    <Class-Name>com.hp.ov.activator.mwfm.component.builtin.JavaRule</Class-Name>
    <Param name="expression" value="x == 1"/>
  </Action>
  <True-Next-Node> PutMessageTrue</True-Next-Node>
  <False-Next-Node>PutMessageFalse</False-Next-Node>
</Rule-Node>
```

## JavaSwitch

`com.hp.ov.activator.mwfm.component.builtin.JavaSwitch`

This node allows the workflow manager to provide conditional if-then-else branching of workflow paths based on the value of a switch key.

The switch key can be computed by complex calculation using a Java code contained in a template file, or embedded as a Java string, or a simple Java expression. The computed key can optionally be saved in a case-packet variable.

The JavaSwitch node works in the same way as the Java node. The main difference is that JavaSwitch is a switch node and the computed value of the switch key can only be a String or an Integer. Long, Float and Double return types are stored as an Integer key.

The node returns WfException and workflow execution fails if:

- A runtime exception is returned during code execution although the code has been compiled and run successfully.
- The result of code execution cannot be converted to a Boolean.

The case values that govern the multiple paths from the Switch node are specified using the action parameters case0, case1...caseN. When the JavaSwitch node is connected to another node, the user is prompted to enter the case value that governs this path; this could be a constant or a case-packet variable of type integer or String. The case params are displayed in a drop down option in the "Arrow drawing UI" along with the default option. The user can select either a case param or the default option.

The default path for the JavaSwitch node is mandatory. The case params are optional. A JavaSwitch node can be connected to the same node and each connection is governed by a different case value.

During workflow execution when the JavaSwitch node is processed, the key is evaluated and an attempt is made to find the matching case value. If a match is found then the workflow node for the case path becomes the next node to be processed by the workflow engine. If a match is not found then the workflow node in the default path is chosen.

The node returns WfException and workflow execution fails if:

- The computed value of the switch key is neither an Integer nor a String
- "The data types of the key and the value values are different
- "Two or more case values have the same value

### See Also

- "Java" on page 191 for more information about the parameters and Java code usage

**Table 4-54 JavaSwitch Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>javafile</i>	No	Name of the java template file. If used as <file>::<method>, then the method specified is called instead of the one specified by the method parameter.	None	String
<i>javacode</i>	No	Java functional declaration.	None	String
<i>method</i>	No	Name of the method to call.	None	String
<i>expression</i>	No	Java expression. Expression can be combined with javafile/javacode and can be wrapped in an extra dummy method.	None	String
<i>arg0...argN</i>	No	Parameters or arguments to pass to the method; cannot be combined with expression.	None	Depends on the method argument type
<i>strict_scope</i>	No	This Java node adheres to scoping rules(As explained above)	True	Boolean
<i>in_scope</i>		List of variables brought into scope if strict_scope is true.	None	String
<i>computed_key</i>	No	Optional variable to capture the computed value of switch key	None	Object
<i>case0...caseN</i>	No	New value to set for the variable. It can be a case-packet variable or a constant (specified as constant:X where X is the constant).	None	Depends on the variable type.
<i>use_solution_dir</i>	No	When set to "true", the nodes will read from \$SOLUTION_ETC/template_files instead of \$ACTIVATOR_ETC/template_files.	false	Boolean

**Example 4-49**      **JavaSwitch - using a Java expression to compute the switch key**

```
<Switch-Node>
  <Name>JavaSwitch</Name>
  <Action>
    <Class-Name>com.hp.ov.activator.mwfm.component.builtin.JavaSwitch</Class-Name>
    <Param name="computed_key" value="computedKey"/>
    <Param name="expression" value="constant:operand1+operan2"/>
    <Param name="strict_scope" value="false"/>
    <Param name="case0" value="add"/>
    <Param name="case1" value="multiply"/>
    <Param name="case2" value="sleep"/>
  </Action>
  <Switch name="case0">Add</Switch>
  <Switch name="case1">Multiply</Switch>
  <Switch name="case2">Sleep</Switch>
  <Switch name="default">DoNothing</Switch>
</Switch-Node>
```



## KillJob

`com.hp.ov.activator.mwfm.component.builtin.KillJob`

The node is used to end a workflow.

**Table 4-55 KillJob Parameters**

Name	Required	Description	Default	Type
<i>job_id</i>	Yes	Name of the variable that contains the identifier of the workflow that you want to stop. The variable must be of the Integer type.	None	Integer

**Example 4-50 KillJob - use in the workflow**

The identifier of the flow to terminate is in the `wf` variable.

```
<Process-Node>
  <Name>End flow</Name>
  <Description>Ends a specified flow</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.KillJob
    </Class-Name>
    <Param name="job_id" value="wf"/>
  </Action>
</Process-Node>

. . .

<Case-Packet>
  <Variable name="wf" type="Integer"/>
</Case-Packet>
```

## LessThan

`com.hp.ov.activator.mwfm.component.builtin.LessThan`

The node allows you to establish whether a variable or a constant is strictly smaller than another. It works with all types of variables. If two variables are of different types, they are compared like strings.

**Table 4-56** LessThan Parameters

Name	Required	Description	Default	Type
<i>op1</i>	Yes	The two parameters are variables or constants. Constant is specified as <code>constant:X</code> . If the two variables are not of the same type, their values are converted into strings and they are compared lexically.	None	Numeric
<i>op2</i>	Yes	Same as above.	None	Numeric

**Example 4-51** LessThan - use in the workflow

It determines whether the value of *var1* is strictly smaller than 0.

```
<Rule-Node disablePersistence="true">
  <Name>Less than?</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.LessThan
    </Class-Name>
    <Param name="op1" value="var1"/>
    <Param name="op2" value="constant:0"/>
  </Action>
  <True-Next-Node>Strictly smaller</True-Next-Node>
  <False-Next-Node>Greater or equal</False-Next-Node>
</Rule-Node>
```

## LessThanOrEqual

`com.hp.ov.activator.mwfm.component.builtin.LessThanOrEqual`

The node allows you to establish whether a variable or a constant is smaller than or equal to another. It works with all types of variables. If two variables are of different types, they are compared like strings.

**Table 4-57** **LessThanOrEqual Parameters**

Name	Required	Description	Default	Type
<i>op1</i>	Yes	The two parameters are variables or constants. Constant is specified as <code>constant:X</code> . If the two variables are not of the same type, their values are converted into strings and they are compared lexically.	None	Numeric
<i>op2</i>	Yes	Same as above.	None	Numeric

**Example 4-52** **LessThanOrEqual - use in the workflow**

This example determines whether the value of *var1* is smaller than or equal to 0.

```
<Rule-Node disablePersistence="true">
  <Name>Less than or equal?</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.LessThanOrEqual
    </Class-Name>
    <Param name="op1" value="var1"/>
    <Param name="op2" value="constant:0"/>
  </Action>
  <True-Next-Node> Smaller or equal</True-Next-Node>
  <False-Next-Node>Strictly smaller</False-Next-Node>
</Rule-Node>

. . .

<Case-Packet>
  <Variable name="var1" type="Integer"/>
</Case-Packet>
```

## Log

`com.hp.ov.activator.mwfm.component.builtin.Log`

The node allows you to log an entry in a log file from a workflow. The output log can be seen from the Logs page in the UI, under the respective module name.

**Table 4-58 Log Parameters**

Name	Required	Description	Default	Type
<i>component_name</i>	Yes	Name of the component logging the message.	None	String
<i>service_id</i>	No	This parameter is used during Solution Logging. The default value is the system case-packet <code>SERVICE_ID</code> .	<code>SERVICE_ID</code>	String
<i>log_level</i>	Yes	Indicates Logging levels. The value should be <code>DEBUG</code> , <code>DEBUG2</code> , <code>INFORMATIVE</code> , <code>WARNING</code> , or <code>ERROR</code> .	None	String
<i>log_message</i>	Yes	The message to be logged.	None	String
<i>param0</i> , <i>param1</i> , ... <i>paramN</i>	No	Parameters to replace free variables in the message.	None	String
<i>part_name</i>	No	Indicates the name of the part logging the message. The value can be either <code>FRAMEWORK</code> or <code>COMPONENT</code> .	<code>COMPONENT</code>	String
<i>topic_name</i>	No	Indicates the name of the topic for the message. The value can be <code>NO_TOPIC</code> , <code>TOPIC_STATISTICS</code> , <code>TOPIC_STARTUP</code> , <code>TOPIC_RECOVERY</code> , <code>TOPIC_COMMON_OPERATION</code> , or <code>TOPIC_SHUTDOWN</code> .	<code>NO_TOPIC</code>	String
<i>log_manager</i>	No	Name of the log module used for storing and accessing the logs.	<code>log_manager</code>	String

**Example 4-53 Log - use in the workflow**

```
<Process-Node>
  <Name>Log</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Log</Class-Name>
    <Param name="component_name" value="Log Node"/>
    <Param name="log_level" value="INFORMATIVE"/>
    <Param name="log_message" value="From MWFM lognode"/>
    <Param name="part_name" value="FRAMEWORK"/>
    <Param name="topic_name" value="NO_TOPIC"/>
  </Action>
```

`</Process-Node>`

## MapData

`com.hp.ov.activator.mwfm.component.builtin.MapData`

This node is a general purpose node that is used to extract data from Maps (for example, HashMaps). You can use it in conjunction with the `Activate` node to extract data uploaded from activation, or you can use it to extract data from your own Map. The `MapData` node is very similar in functionality to the `map` syntax in the Workflow Manager (see “Maps” on page 44). The primary difference between the two is that the `MapData` node allows keys to be embedded with variables. For example, `machine%num%` is a valid key.

The `map_var` parameter points to a previously constructed Map. To extract the data from a Map that was uploaded during a task activation, for example, set the value of the `map_var` parameter to the value of the `uploaded_data_var` parameter that was populated by the `Activate` node.

The names of the subsequent parameters of the `MapData` node indicate the names of case-packet variables that will be written to with data extracted from the Map. The values of these parameters indicate the keys that will be used to extract data from the Map. You can specify one or more case-packet variable parameters.

---

**NOTE** If a requested key is not found in the Map, the `MapData` node will throw a `WFException`.

**Table 4-59** **MapData Parameters**

Name	Required	Description	Default	Type
<code>map_var</code>	Yes	Case-packet variable of type Object that contains the map.	None	Object
<code>name of a case-packet variable</code>	Yes	The name provided is the name of a case-packet variable. The value provided is a key into the map indicated by <code>map_var</code> . The call to <code>MapData</code> sets the value of the named case-packet variable to the value of the map entry associated with this key.  You can specify one or more case-packet variable parameters.	None	

**Example 4-54**      **MapData - use in the workflow**

This example assumes that the map “my\_map” contains a key “db2” with the value “db2.domain.com” and a second key “web2” with the value “web2.domain.com.” The case-packet variable “num” is set to the value 2, and two case-packet variables of type String are defined with the names “webServerMachine” and “dbServerMachine.” The following node entry sets the case-packet variable “webServerMachine” to have the value “web2.domain.com” and the case-packet variable “dbServerMachine” to have the value “db2.domain.com”.

```
<Process-Node disablePersistence="true">
  <Name>Map Data</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.MapData
    </Class-Name>
    <Param name="map_var" value="my_map" />
    <Param name="dbServerMachine" value="db%num%" />
    <Param name="webServerMachine" value="web%num%" />
  </Action>
</Process-Node>
```

---

**NOTE**

Example 4-54 uses keys and values that are of type String. Any Object type can be used for the key and value, however. The key must correctly override the method `Object.equals()` and, if the Map is a `HashMap`, the method `Object.hashCode()`.

---

## MatchDBQuery

`com.hp.ov.activator.mwfm.component.builtin.MatchDBQuery`

The node provides means to use the results of a database query to set the values of multiple case-packet variables in one query. The query is expected to return two columns of data where each row returned from the query sets the value of one case-packet variable. The columns in the row indicate which variable to set and what value to set. Thus, the query should return only two columns.

**Table 4-60 MatchDBQuery Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Specifies the database module to use in order to perform the query.	"db"	String
<i>query</i>	Yes	Query to be carried out. Query must only select two columns. One column is matched to the attribute name and the other to its value.	None	String
<i>attribute_name</i>	No	The name of the column to be used as the attribute name.	"name"	String
<i>attribute_value</i>	No	The name of the column to be used as the attribute value.	"value"	String
<i>param0, param1...paramN</i>	Yes	A series of these parameters can be used to specify the values for free variables (question marks) in the statement.	None	String
<i>no_error</i>	No	If a row is selected but it is not in the case-packet then no exception is thrown if <i>no_error</i> is set to true.	"false"	Boolean

**Example 4-55 MatchDBQuery - use in the workflow**

This example does not use free variables in the SQL query.

```
<Process-Node disablePersistence="true">
  <Name>Read table</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.MatchDBQuery
    </Class-Name>
    <Param name="db" value="db"/>
    <Param name="query" value="select name, value from mytable"/>
    <Param name="attribute_name" value="name"/>
    <Param name="attribute_value" value="value"/>
  </Action>
</Process-Node>
```



**Example 4-56 MatchDBQuery - using free variables in the query**

This example shows the use of free variables in the SQL query. In this case, the value for the free variable is the one held in the `myvar` case-packet variable.

```
<Process-Node disablePersistence="true">
  <Name>Read table 2</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.MatchDBQuery
    </Class-Name>
    <Param name="db" value="db"/>
    <Param name="query" value="select name, value from
      mytable where cust_name= ?"/>
    <Param name="attr_name" value="name"/>
    <Param name="attr_value" value="value"/>
    <Param name="param0" value="myvar"/>
  </Action>
</Process-Node>
...
<Case-Packet>
  <Variable name="myvar" type="String"/>
...
</Case-Packet>
...
```

## MatchDBStore

`com.hp.ov.activator.mwfm.component.builtin.MatchDBStore`

The node provides means to store some or all of the case-packet contents into a database table. An SQL statement is run once for each variable to be stored. Thus, the statement should be an insert or update statement.

The statement must provide at least two free variables (typically the last two) to insert each attribute name and its value. The other free variables (if any) can be used for any purpose. If the free variables are to be used for the name and value of the variables, do not occupy the last two positions in the statement, two additional parameters are supported to indicate their position. Use `attr_name_col` and `attr_value_col` for this purpose.

**Table 4-61 MatchDBStore Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Specifies the database module to use in order to perform the statement.	"db"	String
<i>statement</i>	Yes	SQL statement to be carried out for each variable.	None	String
<i>attr_name_col</i>	No	Indicates the number of the free variable in the SQL statement that should be replaced by the name of the case-packet variable being stored. Note: Use the value 1, not 0, to refer to the first free variable in the SQL statement.	None	Numeric
<i>attr_value_col</i>	No	Indicates the number of the free variable in the SQL statement that should be replaced by the value of the case-packet variable being stored. Note: Use the value 1, not 0, to refer to the first free variable in the SQL statement.	None	Numeric
<i>param0</i> , <i>param1</i> ... <i>paramN</i>	No	A series of these parameters can be used to specify the values for free variables (question marks) in the statement.	None	String
<i>variable0</i> , <i>variable1</i> ... <i>variableN</i> .	Yes	List of case-packet variables to be stored. There can be as many variable parameters as needed.	None	String

**Example 4-57 MatchDBStore - use in the workflow**

This example stores the case-packet variables `username` and `password` into a database table called `footable`. These variables have an additional flag in the database with the same value as the one that the state variables had when the SQL statement was run.

```
<Process-Node disablePersistence="true">
  <Name>Store variables</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.MatchDBStore
    </Class-Name>
    <Param name="db" value="db"/>
    <Param name="statement" value="insert into footable (name,
      value, customer_id) values(?, ?, ?)"/>
    <Param name="attr_name_col" value="1"/>
    <Param name="attr_value_col" value="2"/>
    <Param name="param0" value="customer_id"/>
    <Param name="variable0" value="username"/>
    <Param name="variable1" value="password"/>
  </Action>
</Process-Node>
...
<Case-Packet>
...
  <Variable name="customer_id" type="String"/>
  <Variable name="username" type="String"/>
  <Variable name="password" type="String"/>
...
</Case-Packet>
```

Assuming the case-packet variables have the following values, the statement will insert two new rows into the table as shown:

```
customer_id = 501
username = "AlphaGraphics"
password = "designer6"
```

username	AlphaGraphics	501
password	designer6	501

## MethodRule

`com.hp.ov.activator.mwfm.component.builtin.MethodRule`

The `MethodRule` node works in the same way as the `InvokeMethod` node. The difference is that `MethodRule` is a rule node, which will branch according to the result of the invoked method. The result is interpreted as a boolean according to the rule defined for the return-value of the `InvokeMethod`. If the method throws an exception, and the `throw_except` argument is false, then the rule will follow the false-arrow of the rule in the workflow.

**Table 4-62 MethodRule Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Name of the database module to be used if the method takes a <code>Connection</code> argument.	"db"	String
<i>bean</i>	Yes	A case-packet variable containing the object on which to invoke the method. Alternatively, its value can begin with <code>constant:</code> followed by the fully qualified name of the class on which to invoke a static method	None	String/ Object
<i>arg0</i> <i>arg1...</i> <i>argN</i>	No	Arguments to be passed to the method. If its value begins with <code>constant:</code> the key is value provided after this. Otherwise, the key indicates the name of a variable that holds the key value. If the method takes a parameter of type <code>java.sql.Connection</code> , it is passed implicitly, and should not be listed among the <code>arg</code> parameters to <code>InvokeMethod</code>	None	Any
<i>variable</i>	Yes	Case-packet variable holding the invoked method's return value.	None	Any
<i>method</i>	Yes	Name of the method to be invoked.	None	String

## ModifyScheduledJob

`com.hp.ov.activator.mwfm.component.builtin.ModifyScheduledJob`

The node allows you to modify a scheduled job from inside the workflow. It works similarly to the `ScheduleJob` node. One of the node parameters is the ID of the scheduled job which must be modified. If the ID is not found on the list of scheduled jobs, you receive an error message that the specified scheduled job does not exist.

If the node finishes without errors, the `RET_VALUE` case-packet variable is set to 0. In case of any error in the node, `RET_VALUE` is set to 1. `RET_TEXT` holds information about the exception.

If you set the `throw_excep` parameter to “false”, the node finishes normally even if the specified scheduled job ID does not exist on the list of scheduled jobs. However, `RET_VALUE` is set to 1 and `RET_TEXT` contains information that the specified scheduled job does not exist on the scheduled jobs list.

### See Also

- “ScheduleJob” on page 286.

**Table 4-63** **ModifyScheduledJob Parameters**

Name	Required	Description	Default	Type
<i>scheduled_job_id</i>	Yes	The ID of the scheduled job you want to modify.	None	Integer
<i>schedule_time</i>	No	The date and the time to start the workflow. This parameter accepts date and time as milliseconds with the value starting from January 1, 1970:00:00:00:000. The value should be numeric.	current time	Integer
<i>group_id</i>	No	Used to group a set of scheduled jobs. Also used in connection with timed services and reoccurring scheduled workflows where a common identifier is needed.	None	String

**Table 4-63 ModifyScheduledJob Parameters (Continued)**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>reoccurrence_freq</i>	No	Reoccurrence frequency period. This parameter has to be specified if the scheduled job has to be run repeatedly. If <i>reoccurrence_freq</i> is not specified, the parameter accepts the value as seconds. The value should be numeric.	0	Integer
<i>reoccurrence_end_time</i>	No	The time when reoccurrence of schedule job must end. This parameter accepts the date and time as milliseconds with the value starting value from January 1, 1970 00:00.00:000 GMT. The value should be numeric.	0	Integer
<i>description</i>	No	Description of a scheduled job. The default value is an empty string.	None	String
<i>status</i>	No	Status of a scheduled job. The default value is an empty string.	None	String
<i>reoccurrence_freq_units</i>	No	Reoccurrence frequency units: 1-second, 2 minutes, 3-hours, 4-days, 5-weeks, 6-months. The default value is 1-second. The value should be numeric and can start from 1 to 6.	None	Integer

## MoveFile

`com.hp.ov.activator.mwfm.component.builtin.MoveFile`

The node moves or renames a file.

**Table 4-64**      **MoveFile Parameters**

Name	Required	Description	Default	Type
<i>file</i>	Yes	The name of the file you want to move. You can specify a case-packet variable or a constant. If the path is relative, it is interpreted as relative to <code>\$ACTIVATOR_VAR</code>	None	String
<i>destination</i>	Yes	The destination of the moved file. You can specify a case-packet variable or a constant. This can be a new file name or a directory. If the path is relative, it will be interpreted as relative to <code>\$ACTIVATOR_VAR</code>	None	String

## MultiAssign

`com.hp.ov.activator.mwfm.component.builtin.MultiAssign`

The node is a component used for assigning values to case-packet variables.

### See Also

- “VariableMapper” on page 323
- “Assign” on page 109

**Table 4-65 MultiAssign Parameters**

Name	Required	Description	Default	Type
<i>variable0</i> <i>variable1</i> ... <i>variableN</i>	Yes	Case-packet variables to be set.	None	String / Integer / Float / Boolean / Object
<i>value0</i> <i>value1...</i> <i>valueN</i>	Yes	New value to set for the variable. It can be a case-packet variable or a constant (specified as <code>constant:X</code> where <i>X</i> is the constant).	None	Depends on the variable type.

**Example 4-58 MultiAssign - use in the workflow**

This example sets the counter variable to a value of 0 and the operator value to the value brown.

```
<Process-Node disablePersistence="true">
  <Name>Reset the counter</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.MultiAssign
    </Class-Name>
    <Param name="variable0" name="counter" />
    <Param name="value0" name="constant:0" />
    <Param name="variable1" name="operator" />
    <Param name="value1" name="constant:brown" />
  </Action>
</Process-Node>
```



## Multiply

`com.hp.ov.activator.mwfm.component.builtin.Multiply`

The node multiplies two values.

**Table 4-66** Multiply Parameters

Name	Required	Description	Default	Type
<i>op0</i>	Yes	Name of a case-packet variable that is the first variable to be multiplied. This is also the variable in which the result is saved.	None	Numeric
<i>op1</i>	Yes	Name of a case-packet variable that is the second variable to be multiplied. A hard-coded value can be specified using <code>constant:X</code> where <i>X</i> is the constant value.	None	Numeric

**Example 4-59** Multiply - use in the workflow

This example demonstrates an operation comparable to the statement `x = x * 10` in a language such as Java.

```
<Process-Node disablePersistence="true">
  <Name>Multiply by a factor of 10</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Multiply
    </Class-Name>
    <Param name="op0" value="x"/>
    <Param name="op1" value="constant:10"/>
  </Action>
</Process-Node>
```

## MutexGetInfo

`com.hp.ov.activator.mwfm.component.builtin.conflict.MutexGetInfo`

The node will return the info string of the mutex specified by solution, key, value parameters.

The node will fail if the mutex is not locked.

The mutex workflow nodes distinguish between if the workflow is configured to use persistence or not and it is not possible to combine workflows which are using persistence with workflow which are not using persistence when using the mutex workflow nodes. There are no check for this in the workflow nodes so it is up to the workflow developer to ensure this.

To use the `MutexGetInfo` workflow node the `conflict_module` must be configured.

### See Also

- “MutexLock” on page 219.
- “MutexUnlock” on page 222.
- “MutexSetInfo” on page 221.
- “ConflictModule” on page 373

A text string the job wants to be possible for other jobs to read while the job has locked the mutex.

**Table 4-67**

### MutexGetInfo Parameters

Name	Required	Description	Default	Type
<i>solution</i>	No	The solution name	None	String
<i>key</i>	Yes	The key name	None	String
<i>value</i>	Yes	The value for the key	None	String
<i>info_string</i>	No	A case-packet variable where the info string should be returned in	None	String

## MutexLock

`com.hp.ov.activator.mwfm.component.builtin.conflict.MutexLock`

The node will lock the mutex specified by the combination of solution, key, and value parameters. If the mutex is already locked by another job then the workflow node will wait until the other job has unlocked the mutex if the `wait` parameter is set to `true`. If the `wait` parameter is set to `false` then the node will return immediately. The parameters `locked_job_id` and `locked_info_string` can be used to get information about which job right now has the mutex.

The mutex will automatically be released when the job ends, but the workflow can also call the `MutexUnlock` workflow node if the workflow wants to release the mutex before it ends.

The mutex workflow nodes distinguish between if the workflow is configured to use persistence or not and it is not possible to combine workflows which are using persistence with workflow which are not using persistence when using the mutex workflow nodes. There are no check for this in the workflow nodes so it is up to the workflow developer to ensure this.

If a workflow which has a mutex is scheduled by means of the workflow node `ScheduleCurrentJob` then if the workflow is not using persistence the mutex will be released. If the job is using persistence then the mutex will stay locked during the scheduling.

A mutex is locked across all cluster nodes and if a failover of a job happens the mutex will still be locked when started again on another cluster node.

If the workflow is using persistence then when a mutex is released then the job with the highest priority will get the lock. If more than one workflow is waiting on the mutex and have the same priority then the job which requested the job first will get the lock. To determine which job requested the mutex first the current time on the different cluster nodes will be used.

It is possible to lock more than one mutex in the same job. But if the same mutex is tried to be locked more than one time the workflow node will fail.

To use the `MutexLock` workflow node the `conflict_module` must be configured.

### See Also

- “MutexGetInfo” on page 218.
- “ConflictModule” on page 373

**Table 4-68**      **MutexLock Parameters**

Name	Required	Description	Default	Type
<i>solution</i>	No	The solution name	None	String
<i>key</i>	Yes	The key name	None	String
<i>value</i>	Yes	The value for the key	None	String

**Table 4-68**      **MutexLock Parameters (Continued)**

Name	Required	Description	Default	Type
<i>info_string</i>	No	A text string the job wants to be possible for other jobs to read while the job has locked the mutex.	None	String
<i>wait</i>	No	Indicating if the node should wait until mutex is freed if already locked by another job	None	Boolean
<i>result</i>	Yes, if wait is set to true	The Case-packet which should contain the result of if the mutex has been locked or not. The case-packet will be set to true if the mutex has been locked and else to false.	None	Boolean
<i>locked_job_id</i>	No	The id of the job currently holding the lock.	None	Integer
<i>locked_info_string</i>	No	The info string value of the job currently holding the lock.	None	String

**Example 4-60**      **Mutex - use in the workflow**

This example how a mutex can be locked where the workflow node will be waiting until the mutex is free if already used by another job.

```

<Process-Node>
  <Name>Lock the mutex</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.conflict.MutexLock
    </Class-Name>
    <Param name="info_string" value="I have the lock"/>
    <Param name="key" value="constant:ABC"/>
    <Param name="solution" value="constant:mutex"/>
    <Param name="value" value="constant:12345"/>
  </Action>
</Process-Node>

```

## MutexSetInfo

`com.hp.ov.activator.mwfm.component.builtin.conflict.MutexSetInfo`

The node will set the info string of the mutex specified by solution, key, value parameters if the mutex is already locked by this job.

The node will fail if the mutex is not locked.

The mutex workflow nodes distinguish between if the workflow is configured to use persistence or not and it is not possible to combine workflows which are using persistence with workflow which are not using persistence when using the mutex workflow nodes. There are no check for this in the workflow nodes so it is up to the workflow developer to ensure this.

To use the `MutexSetInfo` workflow node the `conflict_module` must be configured.

### See Also

- “MutexUnlock” on page 222.
- “MutexGetInfo” on page 218.
- “ConflictModule” on page 373

**Table 4-69**

### MutexSetInfo Parameters

Name	Required	Description	Default	Type
<i>solution</i>	No	The solution name	None	String
<i>key</i>	Yes	The key name	None	String
<i>value</i>	Yes	The value for the key	None	String
<i>info_string</i>	No	A text string the job wants to be possible for other jobs to read while the job has locked the mutex.	None	String

## MutexUnlock

`com.hp.ov.activator.mwfm.component.builtin.conflict.MutexUnlock`

The node will unlock the mutex specified by the combination of solution, key, and value parameters. If the mutex is not locked or locked by another job then the workflow node will fail.

The mutex workflow nodes distinguish between if the workflow is configured to use persistence or not and it is not possible to combine workflows which are using persistence with workflow which are not using persistence when using the mutex workflow nodes. There are no check for this in the workflow nodes so it is up to the workflow developer to ensure this.

To use the `MutexUnlock` workflow node the `conflict_module` must be configured.

### See Also

- “MutexLock” on page 219.
- “ConflictModule” on page 373

**Table 4-70**      **MutexUnlock Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>solution</i>	No	The solution name	None	String
<i>key</i>	Yes	The key name	None	String
<i>value</i>	Yes	The value for the key	None	String

## NAddConfigurationPolicy

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAddConfigurationPolicy`

The node adds a policy and a set of configuration rules to a specified device group in NA to check service configuration presence. When a policy is created for each service it is recommended to use a policy-tag such as "HPSA VPN Service". This helps the NA operator to filter numerous policies in the NA GUI .

If the `event_rule` parameter is specified, an event rule will be assigned to the policy which will launch a dummy workflow on violation.

The `rule_list` parameter is an array of Maps with the following keys:

- `start_pattern`: The pattern defining where the text block starts for condition checking as a String.
- `end_pattern`: The pattern defining where the text block ends for condition checking as a String.
- `name`: The rule name as a String. This is optional, if the provided Map does not contain the 'name' key the rule name will be set to `<policy_name><rule_number>`.
- `conditions`: A String array with the condition values. The `NABuildConditionList` node may be used to generate one.

This parameter can be generated dynamically by a Java node or by using `NABuildRuleList` and `NABuildConditionList` nodes.

The new policy id is returned in `result_var` as a String.

**Table 4-71**

### NAddConfigurationPolicy Parameters

Name	Required	Description	Default	Type
<code>module_name</code>	Yes	The name of the <code>NARequestModule</code> to be used.	None	String
<code>result_var</code>	Yes	The case packet variable where the result will be stored.	None	String
<code>policy_name</code>	Yes	The name of the policy to be created.	None	String
<code>policy_description</code>	No	A human readable description for the policy to be created.	None	String
<code>policy_tag</code>	No	Adds a tag to the policy. The policy tag helps the operator filter the policies in the GUI.	None	String
<code>group_name</code>	Yes	The name of the device group the policy should be applied to.  If the group does not exist, the group will be created.	None	String

**Table 4-71**      **NAAddConfigurationPolicy Parameters (Continued)**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>rule_list</i>	No	Rule definitions to be added to the policy. The <code>NABuildRuleList</code> workflow node can be used to generate such rule definitions.	None	Object (Map[])
<i>event_rule</i>	No	Name of the event rule the policy should be associated to. If this parameter is not present, the policy will not be added to any event rule.	None	String



## NAAddDevice

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAAddDevice`

The node adds a device to NA.

**Table 4-72** NAAddDevice Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>comment</i>	No	Additional information regarding the device.	None	String
<i>consoleip</i>	No	Device console IP.	None	String
<i>consoleport</i>	No	Device console port number.	None	String
<i>description</i>	No	The descriptive name of the device	None	String
<i>hostname</i>	Yes, if ip is not used	Device host name.	None	String
<i>ip</i>	Yes, if hostname is not used	Device IP.	None	String
<i>model</i>	No	Device model.	None	String
<i>vendor</i>	No	Device vendor.	None	String

## NAAddDeviceGroup

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAAddDeviceGroup`

The node can be used to add a device group to NA.

If no `parent_group` is specified the device group will be the top parent group.

**Table 4-73** NAAddDeviceGroup Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>group_name</i>	Yes	The name of the device group to be created.	None	String
<i>parent_group_name</i>	No	The <code>parent_group</code> for the group. If the group does not exist it will be created. If no <code>parent_group</code> is specified the device group will be the top parent group.	top parent group	String
<i>comment</i>	No	Optional device group comment.	None	String

## NAAddDeviceToGroup

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAAddDeviceToGroup`

The node adds a device to a specify device group in NA.

**Table 4-74** NAAddDeviceToGroup Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>device_id</i>	Yes	The device identifier.	None	String
<i>group_name</i>	Yes	The device group name.	None	String

## NAAddRuleToPolicy

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAAddRuleToPolicy`

Adds a rule and the associated set of conditions to an existing policy. The NA type of rules being created is Configuration. For each rule a configuration block is defined. A configuration block is defined by its start and end delimiters, and one or more conditions are defined for each one. A condition is a regular expression defining the portion of text that must be present inside the corresponding block. When several conditions are specified for a rule (configuration block), a Boolean expression is used to calculate the result of the check..

**Table 4-75 NAAddRuleToPolicy Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>policy_id</i>	Yes, if <i>policy_name</i> is not used	The NA id of the policy to be modified.	None	String
<i>start_pattern</i>	Yes	The pattern defining where the text block start for the checking of the conditions.	None	String
<i>end_pattern</i>	Yes	The pattern defining where the text blocks end for the checking of the conditions.	None	String
<i>rule_name</i>	Yes	The name for the new rule.	None	String
<i>conditions</i>	Yes	The regular expressions defining the text that must be present inside the blocks. All conditions for a rule will be evaluated with the "AND" operator in NA. The node <code>NABuildConditionList</code> can be used to generate these condition lists.	None	String

## NABuildConditionList

`com.hp.ov.activator.mwfm.component.builtin.narequest.NABuildConditionList`

The node takes a variable number of `Strings` as input and produces a `String` array from them that can be used as a condition list for the `BuildRuleList` operation on NA.

**Table 4-76** NABuildConditionList Parameters

Name	Required	Description	Default	Type
<i>result_var</i>	Yes	The case packet variable where the result will be stored	None	Object
<i>condition 0,</i> <i>condition 1,</i> <i>...</i> <i>condition N</i>	Yes, at least one	The regular expressions defining the text that must be present inside the blocks. All conditions for a rule will be evaluated with the "AND" operator in NA	None	String

## NABuildRuleList

`com.hp.ov.activator.mwfm.component.builtin.narequest.NABuildRuleList`

The node takes a variable number of rule information and produces a list of maps containing rule information to be used as input for the `rule_list` parameter of `NAAddConfigurationPolicy`. For each rule a configuration block is defined. A configuration block is defined by its start and end delimiters, and one or more conditions are defined for each one. A condition is a regular expression defining the portion of text that must be present inside the corresponding block. When several conditions are specified for a rule (configuration block), a Boolean AND expression is used to calculate the result of the check. The NA type of rules being created is `Configuration..`

**Table 4-77 NABuildRuleList Parameters**

Name	Required	Description	Default	Type
<i>result_var</i>	Yes	The case packet variable where the result will be stored	None	Object
<i>start_pattern0,</i> <i>start_pattern1,</i> ... <i>start_patternN</i>	Yes, at least one	The patterns defining where the text blocks start for the checking of the conditions.	None	String
<i>end_pattern0,</i> <i>end_pattern1,</i> ... <i>end_patternN</i>	Yes, at least one	The patterns defining where the text blocks end for the checking of the conditions. The number of end patterns must match the number of start patterns.	None	String
<i>name0,</i> <i>name1,</i> ... <i>nameN</i>	No	The names for the rules to be created. A rule is created for each <start/end>_pattern pair regardless of the presence of this parameter (see <code>AddConfigurationPolicy</code> API for the default value for rule names when this is undefined).	None	String
<i>condition_s0,</i> <i>condition_s1,</i> ... <i>condition_sN</i>	Yes, at least one	The regular expressions defining the text that must be present inside the blocks. All conditions for a rule will be evaluated with the "AND" operator in NA	None	Object (Array of Strings)

## NADeleteDeviceGroup

`com.hp.ov.activator.mwfm.component.builtin.narequest.NADeleteDeviceGroup`

The node deletes a device group in NA.

**Table 4-78** NADeleteDeviceGroup Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>group_name</i>	Yes	The name of the device group to be deleted.	None	String

## NADeletePolicy

`com.hp.ov.activator.mwfm.component.builtin.narequest.NADeletePolicy`

The node deletes a policy in NA.

**Table 4-79** NADeletePolicy Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NARequestModule to be used.	None	String
<i>policy_id</i>	Yes	The NA id of the policy to be deleted.	None	String



## NAGetSnapshot

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAGetSnapshot`

The node takes a snapshot of the specified device in NA.

**Table 4-80 NAGetSnapshot Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>result_var</i>	Yes	The case packet variable where the snapshot will be stored.	None	String
<i>device_id</i>	Yes	Device identifier.	None	String
<i>sync</i>	No	Return only after the snapshot retrieval task is complete. Otherwise return immediately after the task is scheduled.	false	Boolean
<i>comment</i>	No	Optional comment for the snapshot.	None	String

## NAListConfigId

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAListConfigId`

The node list all config identifiers present in NA unless any of the options to limit the listing are specified.

**Table 4-81** NAListConfigId Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NARquestModule to be used.	None	String
<i>result_var</i>	Yes	The case packet variable where the result will be stored. The result will be a list of identifiers returned as String[].	None	Object
<i>device_id</i>	Yes	Device identifier.	None	String
<i>start</i>	No	Display only those configs stored on or after the given date.	None	String
<i>end</i>	No	Display only those configs stored on or before the given date.	None	String

## NAListDevice

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAListDevice`

Lists all device identifiers present in NA unless any of the options to limit the listing are specified.

**Table 4-82 NAListDevice Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>result_var</i>	Yes	The case packet variable where the result will be stored. The result will be a Map[] containing device information as returned by NA.	None	Object
<i>group_name</i>	No	List only devices in this device group.	None	String
<i>id</i>	No	List only the device specified by this identifier.	None	String
<i>hostname</i>	No	List only devices with this host name.	None	String
<i>ip</i>	No	List only devices with this IP Address.	None	String
<i>limitcount</i>	No	Return this many results.	None	String

## NAModifyConditionsOnRule

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAModifyConditionsOnRule`

The node can modify existing conditions on a given policy rule.

If the new condition value is "" or null, the old condition will be deleted. If the old condition value is "" or null, the new condition will be added.

**Table 4-83 NAModifyConditionsOnRule Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>policy_id</i>	Yes, if policy name is not defined	The NA id of the policy to be modified.	None	String
<i>rule_name</i>	No	The name of the rule to be modified.	None	String
<i>old_conditions</i>	Yes	The values of the conditions that must be replaced. NABuildConditionList node can be used to generate these condition lists	None	String[]
<i>new_conditions</i>	Yes	The new values for the conditions. The number of new condition values must match the number of old ones, as each new_condition will replace the corresponding old condition. The NABuildConditionList can be used to generate these condition lists.	None	String[]
<i>regex</i>	No	Specifies if the values in old_conditions should be considered regular expressions for matching.	false	Boolean

## NARemoveDeviceFromGroup

`com.hp.ov.activator.mwfm.component.builtin.narequest.NARemoveDeviceFromGroup`

The node can be used to remove a device from a group on NA.

**Table 4-84** NARemoveDeviceFromGroup Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NARquestModule to be used.	None	String
<i>device_id</i>	Yes	The device identifier.	None	String
<i>group_name</i>	Yes	The device group name.	None	String

## NARemoveRuleFromPolicy

`com.hp.ov.activator.mwfm.component.builtin.narequest.NARemoveRuleFromPolicy`

The node can be used to remove a rule from a policy on NA.

**Table 4-85** NARemoveRuleFromPolicy Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NARequestModule to be used.	None	String
<i>policy_id</i>	Yes, if <i>policy_name</i> is not used	The NA id of the policy to be removed.	None	String
<i>rule_name</i>	Yes	The NA name of the rule to be removed.	None	String

## NARunAdvancedScript

`com.hp.ov.activator.mwfm.component.builtin.narequest.NARunAdvancedScript`

The node runs an existing advanced script against a device or group of ddevices.

**Table 4-86 NARunAdvancedScript Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NARequestModule to be used.	None	String
<i>result_var</i>	Yes	The case packet variable where the script result string will be stored.	None	String
<i>device_id</i>	Yes	The identifier of the device in which the script is to be run on.	None	String
<i>name</i>	Yes	The name of the advanced script to run.	None	String
<i>parameters</i>	No	Command line parameters for the advanced script to run.	None	String
<i>variables</i>	No	A list of variables to be replaced in the script provided as a mapping of name=value pairs.	None	String
<i>sync</i>	No	Return only after the run script task is complete. Otherwise return immediately after the task is scheduled.	false	Boolean
<i>nowait</i>	No	Do not wait if there is another task currently running against the same device.	false	Boolean
<i>comment</i>	No	An optional comment about the script being run.	None	String

## NARunCommandScript

`com.hp.ov.activator.mwfm.component.builtin.narequest.NARunCommandScript`

The node runs an existing command script against a device.

**Table 4-87** NARunCommandScript Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NARequestModule to be used.	None	String
<i>result_var</i>	Yes	The case packet variable where the script result string will be stored.	None	String
<i>device_id</i>	Yes	The identifier of the device in which the script is to be run on.	None	String
<i>name</i>	Yes	The name of the command script to run.	None	String
<i>variables</i>	No	A list of variables to be replaced in the script provided as a mapping of name=value pairs.	None	String
<i>linebyline</i>	No	Indicates that line by line deployment is preferred, rather than file-based deployment.	false	Boolean
<i>sync</i>	No	Return only after the run script task is complete. Otherwise return immediately after the task is scheduled.	false	Boolean
<i>nowait</i>	No	Do not wait if there is another task currently running against the same device.	false	Boolean
<i>comment</i>	No	An optional comment about the script being run.	None	String



## NARunScript

`com.hp.ov.activator.mwfm.component.builtin.narequest.NARunScript`

The node runs a script on a device.

**Table 4-88 NARunScript Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>result_var</i>	Yes	The case packet variable where the script result string will be stored.	None	String
<i>device_id</i>	Yes	The identifier of the device in which the script is to be run on.	None	String
<i>mode</i>	No	A command script mode to run the script in.	None	String
<i>script</i>	Yes	The name of the script that should be run.	None	String
<i>sync</i>	No	Return only after the run script task is complete. Otherwise return immediately after the task is scheduled.	false	Boolean
<i>nowait</i>	No	Do not wait if there is another task currently running against the same device.	false	Boolean
<i>comment</i>	No	An optional comment about the script being run.	None	String

## NAShowConfig

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAShowConfig`

The node can be used to perform a ShowConfig on NA.

**Table 4-89 NAShowConfig Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>result_var</i>	Yes	The name of the case-packet variable where the config text will be stored.	None	String
<i>id</i>	Yes	The identifier of the config to show.	None	String
<i>mask</i>	No	Mask out sensitive information such as device passwords.	None	Boolean

## NAShowDiagnostic

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAShowDiagnostic`

The node can be used to perform a ShowDiagnostic on NA.

**Table 4-90** NAShowDiagnostic Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>result_var</i>	Yes	The name of the case-packet variable where the diagnostic result text will be stored.	None	String
<i>id</i>	Yes	The identifier of the diagnostic which results are to be shown.	None	String

## NAShowTask

`com.hp.ov.activator.mwfm.component.builtin.narequest.NAShowTask`

The node can be used to perform a ShowTask on NA.

**Table 4-91 NAShowTask Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NAResultModule to be used.	None	String
<i>result_var</i>	Yes	The name of the case-packet variable where the task information will be stored.	None	String
<i>id</i>	Yes	The task identifier whose details will be displayed.	None	String

## Not

`com.hp.ov.activator.mwfm.component.builtin.Not`

The node compares a variable or a constant to C programming language style, and returns “false” if the variable or the constant has a value of 0, or “true” in other cases.

**Table 4-92** Not Parameters

Name	Required	Description	Default	Type
<i>op1</i>	Yes	Constant (specified as constant:X), Integer or String variable. If the string is empty, the result is “true”; if the integer is 0, the result is “false.”	None	Any

**Example 4-61**

### Not - use in the workflow

This example verifies whether the value of *var1* is 0.

```
<Rule-Node disablePersistence="true">
  <Name>Not?</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Not
    </Class-Name>
    <Param name="op1" value="var1"/>
  </Action>
  <True-Next-Node>Is zero</True-Next-Node>
  <False-Next-Node>Not zero</False-Next-Node>
</Rule-Node>
```

## PAYG

`com.hp.ov.activator.mwfm.component.builtin.PAYG`

The node PAYG - Pay As You Grow enables to design licensing mechanism for HP Service Activator Solutions.

The node will store the license type (PAYG), Solution Name, Unit (the name of the unit in the solution to be licensed) and Used (the license count) values into the database when invoked.

If `increment` is not set, then the default value is set as 1. If `increment` is set, then it must be a non negative integer value.

A given `solution` and `unit` if defined in PAYG node must not be defined in PPU node.

### See Also

- See “PPU” on page 248

**Table 4-93** PAYG Parameters

Name	Required	Description	Default	Type
<i>unit</i>	Yes	A string indicating the name of a unit in a solution which needs to be licensed	None	String
<i>solution</i>	No	A string indicating the name of a solution which needs to be licensed	None	String
<i>increment</i>	No	The increment count value which needs to be added to the Used count.	1	Integer

### Example 4-62

#### PAYG - use in the workflow

This example creates a database entry for PAYG license type for Solution name MNP and Unit name Port and increments the Used count value by 3 when invoked.

```
<Process-Node>
  <Name>PAYG</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PAYG
    </Class-Name>
    <Param name="unit" value="constant:Port"/>
    <Param name="solution" value="constant:MNP"/>
    <Param name="increment" value="constant:3"/>
  </Action>
</Process-Node>
```

## PatternMatch

`com.hp.ov.activator.mwfm.component.builtin.PatternMatch`

The node matches a string to regular expression pattern. The result is returned in a match variable or in group variables depending on the regular expression. At least if one match is found, the node returns true. If no matches are found, it returns false.

**Table 4-94** **PatternMatch Parameters**

Name	Required	Description	Default	Type
<i>value</i>	Yes	The string to be processed	None	String
<i>pattern</i>	Yes	The regular expression as defined by the <code>java.util.regex.Matcher</code> class.	None	String
<i>startat</i>	No	Variable indicating the index where matching starts. Updated to index after each match.	0	Integer
<i>multiline</i>	No	Pattern spans multiple lines	False	Boolean
<i>nocase</i>	No	Matching case is insensitive	False	Boolean
<i>match</i>	No	Result of the entire match	None	String
<i>group</i>	No	Result of group pattern. Groups are counted from 0.	None	String

## PPU

`com.hp.ov.activator.mwfm.component.builtin.PPU`

The node PPU - Pay Per Use enables to design licensing mechanism for HP Service Activator Solutions.

The node will store the license type (PPU), Solution Name, Unit (the name of the unit in the solution to be licensed) and Used (the license count) values into the database when invoked.

If `increment` is not set, then the default value is set as 1. If `increment` is set, then it must be a non negative integer value.

A given solution and unit if defined in PPU node must not be defined in PAYG node.

### See Also

- See “PAYG” on page 246

**Table 4-95** PAYG Parameters

Name	Required	Description	Default	Type
<i>unit</i>	Yes	A string indicating the name of a unit in a solution which needs to be licensed	None	String
<i>solution</i>	No	A string indicating the name of a solution which needs to be licensed	None	String
<i>increment</i>	No	The increment count value which needs to be added to the Used count.	1	Integer

### Example 4-63 PPU - use in the workflow

This example creates a database entry for PPU license type for Solution name MNP and Unit name Port and increments the Used count value by 3 when invoked.

```
<Process-Node>
  <Name>PAYG</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PAYG
    </Class-Name>
    <Param name="unit" value="constant:MPLS"/>
    <Param name="solution" value="constant:VPM"/>
    <Param name="increment" value="constant:1"/>
  </Action>
</Process-Node>
```



## PutMessage

`com.hp.ov.activator.mwfm.component.builtin.PutMessage`

The node puts a message on a message queue. The messages will be persisted in the database. Optionally, the messages can also be associated with a solution.

**NOTE** If the message is more than 4000 bytes the message will be truncated to 4000 bytes.

**Table 4-96 PutMessage Parameters**

Name	Required	Description	Default	Type
<i>queue</i>	Yes	Queue where the message is left. This parameter can either be a constant or a case-packet variable. Spaces are not allowed.	None	Queue
<i>message</i>	Yes	Message to be printed. Any % s symbols appearing in the string are replaced by consecutive paramN parameters. Functions similar to printf in the C programming language.	None	String
<i>param0, param1, ...</i>	No	If the message contains any % s symbols, the first one is replaced by the value of the variable indicated by param0, param1, and so on. The variables can be of any type. However, their value is converted to a string.	None	Any
<i>service_id</i>	No	The Service Identifier value used for associating the message with a solution.	None	String
<i>order_id</i>	No	The Order Identifier value used for associating the message with a solution.	None	String
<i>type</i>	No	The type value of the workflow.	None	String
<i>state</i>	No	The state value of the workflow.	None	String

**Example 4-64 PutMessage - using a constant queue parameter**

```
<Process-Node>
  <Name>Show message</Name>
  <Description>Shows result</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PutMessage
    </Class-Name>
    <Param name="message" value="The sum is %s"/>
  </Action>
</Process-Node>
```

**Process Nodes, Rule Nodes, and Switch Nodes**

```
        <Param name="param0" value="operand 1"/>
        <Param name="queue" value="sum_queue"/>
    </Action>
</Process-Node>
<Case-Packet>
    <Variable name="operand 1" type="Integer"/>
</Case-Packet>
```

**Example 4-65 PutMessage - using a variable queue parameter**

```
<Process-Node>
  <Name>Show message</Name>
  <Description>Shows result</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PutMessage
    </Class-Name>
    <Param name="message" value="The sum is %s"/>
    <Param name="param0" value="operand 1"/>
    <Param name="queue" value="variable:myqueuevar"/>
  </Action>
</Process-Node>
<Case-Packet>
  <Variable name="operand 1" type="Integer"/>
</Case-Packet>
```

**Example 4-66 PutMessage - using a constant service\_id parameter**

```
<Process-Node>
  <Name>Show message</Name>
  <Description>Shows result</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PutMessage
    </Class-Name>
    <Param name="message" value="The sum is %s"/>
    <Param name="param0" value="operand 1"/>
    <Param name="queue" value="sum_queue"/>
    <Param name="service_id" value="serviceId1"/>
  </Action>
</Process-Node>
<Case-Packet>
  <Variable name="operand 1" type="Integer"/>
</Case-Packet>
```

**Example 4-67 PutMessage - using a variable service\_id parameter**

```
<Process-Node>
  <Name>Show message</Name>
  <Description>Shows result</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PutMessage
    </Class-Name>
    <Param name="message" value="The sum is %s"/>
    <Param name="param0" value="operand 1"/>
    <Param name="queue" value="sum_queue"/>
    <Param name="service_id" value="variable:SERVICE_ID"/>
  </Action>
</Process-Node>
<Case-Packet>
  <Variable name="operand 1" type="Integer"/>
</Case-Packet>
<Initial-Case-Packet>
  <Variable-Value name="SERVICE_ID" value="srvc1"/>
</Initial-Case-Packet>
```

**Example 4-68 PutMessage - using both constant and variable parameters**

```
<Process-Node>
  <Name>Put message</Name>
  <Description>Shows result</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PutMessage
    </Class-Name>
    <Param name="message" value="The param value 0 is constant:%s and the param value
1 is variable test:%s"/>
    <Param name="queue" value="info"/>
    <Param name="param0" value="constant:constantParameter"/>
    <Param name="param1" value="Variable1"/>
  </Action>
</Process-Node>
<Case-Packet>
  <Variable name="Variable1" type="Integer"/>
</Case-Packet>
```

## QueryInventory

`com.hp.ov.activator.mwfm.component.builtin.QueryInventory`

The node used to query the inventory. This node relies on the JavaBeans generated by the InventoryBuilder tool. It sets the `RET_VALUE` to 0 if successful and to 1 if it finds no row. If the `return_array` parameter is set to “true”, the node will always return an array containing zero or more beans. If the parameter `use_cache` is set to true then the result will be saved in the caching module as well as being returned.

---

### NOTE

If the `findBy` method invoked by the `QueryInventory` node returns an array of beans, then the case-packet variable will contain the entire array of beans returned. You can then use the array indexing notation as described under the “Arrays or Vectors” on page 43 in Chapter 2.

---

**Table 4-97** QueryInventory Parameters

Name	Required	Description	Default	Type
<i>db</i>	No	Name of the database module to be used.	“db”	String
<i>bean</i>	Yes	Name of the JavaBean to be used.	None	String
<i>use_cache</i>	No	If set to true then the result will be saved in the configured caching module.	false	Boolean
<i>caching_module</i>	Yes if <code>use_cache</code> is set to true	The name of the caching module to use.	None	String
<i>key_value0</i> , <i>key_value1...</i> <i>key_valueN</i>	Yes	Value of the key by which the element should be looked up. If its value begins with <code>constant</code> : the key is the value provided after this. Otherwise, the key indicates the name of a variable that holds the key value.	None	Object
<i>variable</i>	Yes	Case-packet variable holding the returned bean or bean array.	None	Object
<i>find_by_method</i>	No	Name of the method to use for query.	“findByPrimaryKey”	Integer

**Table 4-97 QueryInventory Parameters (Continued)**

Name	Required	Description	Default	Type
<i>where</i>	No	This parameter appends a simple SQL “WHERE” clause to “SELECT” statement. The parameter may not be used if the <i>find_by_method</i> parameter is set to “findByPrimaryKey”. Any %s symbols appearing in the string are replaced by consecutive paramN parameters.	None	String
<i>preserve</i>	No	If set to true (default), the node will preserve the value of the case-packet variable specified for the ‘variable’ parameter and that the query does not return any beans; otherwise, the node returns ‘null’, if the query does not find any beans. This option is ignored if ‘return_array’ is set to ‘true’. The default value is ‘true’ (VARIABLE or CONSTANT).	True	Object
<i>param0</i> , <i>param1...</i> <i>paramN</i>	No	If the message contains any %s symbols, the first one is replaced by the value of the variable indicated by param0, and so on. The variables can be of any type. However, their values are converted to strings.	None	Any
<i>return_array</i>	No	When set to “true” the node will always return an array containing zero (if the query does not return any beans), or more beans regardless of the return type of the bean method specified by the <i>find_by_method</i> parameter	False	Object

**Example 4-69 QueryInventory - use in the workflow**

This example uses the `QueryInventory` node to retrieve data about a VPN instance previously created with the `InventoryBuilder`. By default, the `findByPrimaryKey` method is used and the VPN instance has its id as the primary key. The resulting bean is saved in the case packet variable `vpn_obj`.

```
<Process-Node disablePersistence="true">
  <Name>Get VPN instance</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.QueryInventory
    </Class-Name>
    <Param name="key_value0" value="vpn_id"/>
    <Param name="variable" value="vpn_obj" />
    <Param name="bean" value="com.hp.ov.activator.example.VPN" />
  </Action>
</Process-Node>

<Case-Packet>
  <Variable name="vpn_obj" type="Object"/>
  <Variable name="vpn_id" type="String"/>
</Case-Packet>
```

## QueryScheduledJob

`com.hp.ov.activator.mwfm.component.builtin.QueryScheduledJob`

The node allows you to query the list of scheduled jobs and get an array of objects (ScheduledJobDescriptor). In HP OVSA 4.1 version, the node was designed in a such that param0, param1, ...paramN parameters was not supporting constant values. However in HP SA 5.1 version, a user can enter constants as parameters. From ScheduledJobDescriptor, you can get details about a scheduled job.

There are two ways to query the list of scheduled jobs. You can use the *scheduled\_job\_id* parameter to specify a job ID and get details about one job. Or you can use the *where* parameter to make your selection criteria less restrictive. In the *where* parameter, you can write a statement using the same structure as in an SQL where clause. You would usually use the *where* parameter when you want to conduct a more complex query. (See the table of the node parameters for more details about using the *where* parameter.) To get a complete list of scheduled jobs, do not supply values for the *scheduled\_job\_id* and “where” attributes. The QueryScheduledJobs node can return an empty array if the list of scheduled jobs is empty.

If the node finishes without errors, the RET\_VALUE case-packet is set to 0. In case of any error in the node, RET\_VALUE is set to 1 and the RET\_TEXT case-packet variable holds more information about the problem. If the job being queried is not in the list of scheduled jobs, the node sets RET\_VALUE to 1, puts the error description in RET\_TEXT, and continues to the next node.

**Table 4-98 QueryScheduledJob Parameters**

Name	Required	Description	Default	Type
<i>scheduled_job_id</i>	No	The ID of the scheduled job for which a SchedulerJobDescriptor must be returned.	0	Integer
<i>ret_result</i>	Yes	Query result in the form of an array of ScheduledJobDescriptor objects is returned to this parameter. If no data is found, query returns an empty array. This parameter must refer to a variable of Object type.	None	Object
<i>where</i>	No	This parameter passes a simple SQL “WHERE” condition statement. In the statement, you can use “%” signs, which mean that parameters will appear in their places.	None	String



**Table 4-98 QueryScheduledJob Parameters (Continued)**

Name	Required	Description	Default	Type
<i>param0</i> , <i>param1</i> , ... <i>paramN</i>	No	The parameters are used as free variables in the “WHERE” statement. The value of the case-packet variable appointed by param0 is replaced with the first occurrence of %s in the <i>where</i> parameter, param1 is replaced with the second and so forth.	None	String
<i>order_by</i>	No	This parameter passes a simple SQL “ORDER BY” statement.	None	String

**Example 4-70 QueryScheduledJobs - use in the workflow**

This example shows how you can get a ScheduledJobDescriptor for a specific scheduled job

```
<Process-Node disablePersistence="true">
  <Name>Query scheduled job by job id</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.QueryScheduledJobs
    </Class-Name>
    <Param name="scheduled_job_id" value="{the scheduled job id}"/>
    <Param name="ret_result" value="result"/>
  </Action>
  <Next-Node>{next node name}</Next-Node>
</Process-Node>
```

After execution of this node the `result` variable contains query the result. To access the result object, use the `VariableMapper` node. In this example we fetch the description (`desc`) for the scheduled job identified by index.

```
<Process-Node disablePersistence="true">
  <Name>Get scheduled job description</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.VariableMapper
    </Class-Name>
    <Param name="in_desc" value="%result[{{index}}].desc%"/>
  </Action>
  <Next-Node>{next node name}</Next-Node>
</Process-Node>
```

**Example 4-71**      **QueryScheduledJobs - use in the workflow**

This example shows how you can get a ScheduledJobDescriptor list for nodes which have no reoccurrence and order the results by job ID in descending order.

```
<Process-Node disablePersistence="true">
  <Name>Query scheduled jobs with where statement</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.QueryScheduledJobs
    </Class-Name>
    <Param name="where" value="constant:REOCCURRING_PERIOD is null"/>
    <Param name="ret_result" value="result"/>
    <Param name="order_by" value="constant:job_id desc"/>
  </Action>
  <Next-Node>{next node name}</Next-Node>
</Process-Node>
```

## QueryServiceInstance

`com.hp.ov.activator.mwfm.component.builtin.QueryServiceInstance`

The node retrieves the values of service-instance parameters. The node expects to have case-packet variables defined for each service-instance parameter stored in the inventory for the specified `service_id`. You can fetch all the service-instance parameters related to the service ID, or you can selectively fetch those of interest. If you do not list any `variable` parameters, then it fetches all the service-instance parameters for the given `service_id`.

If this node fails to find the requested data because the `service_id` specified is not found in the database, then the node sets the `RET_VALUE` case-packet variable to 1.

If the data is fetched from the database, but there is not a case-packet variable to catch the value, then the node continues without an error. A Warning message is logged.

Queries on non-existent service instances return warning messages reading: "Query for service instance parameters returned no data." These messages are placed under the `mwfm_active` tab in the Logs area of the Operator UI.

Error messages are also possible. Usually, they are generated when the definition of any of the node parameters is incorrect. For instance, assigning the wrong value to a variable provokes the following error message:

```
...com.hp.ov.activator.mwfm.component.WFNoSuchAttributeException:
variable "aaa" not present in case packet."
```

In this case "aaa" indicates the value assigned to a variable. Error messages are also located under the `mwfm_active` tab in the Logs area.

**Table 4-99 QueryServiceInstance Parameters**

Name	Required	Description	Default	Type
<code>db</code>	No	Database module to use in order to perform the query.	"db"	String
<code>service_id</code>	Yes	Name of a case-packet variable that holds the unique identifier that the data being queried is tied to. This identifier represents the customer and service that has been activated.	None	String
<code>variable0</code> , <code>variable1...</code> <code>variableN</code>	No	Case-packet variables whose values are to be fetched. You can specify as many <code>variable</code> parameters as necessary. If you do not specify any <code>variable</code> parameters, all service instance parameters are fetched and the node expects case-packet variables to match each service-instance parameter.	None	String

**Example 4-72 QueryServiceInstance - use in the workflow**

This example retrieves several service-instance parameters that are tied to a customer identifier.

```
<Process-Node disablePersistence="true">
  <Name>Technical query inventory</Name>
  <Description>
    Queries the existing information about a customer_id from the service
    instance repository.
  </Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.QueryServiceInstance
    </Class-Name>
    <Param name="db" value="db"/>
    <Param name="service_id" value="customer_id"/>
    <Param name="variable0" value="web_domain"/>
    <Param name="variable1" value="group"/>
    <Param name="variable2" value="homedir"/>
    <Param name="variable3" value="ipaddress"/>
    <Param name="variable4" value="logdir"/>
    <Param name="variable5" value="login"/>
    <Param name="variable6" value="machine"/>
    <Param name="variable7" value="password"/>
    <Param name="variable8" value="port"/>
    <Param name="variable9" value="pre_domain"/>
    <Param name="variable10" value="rootdir"/>
    <Param name="variable11" value="uid"/>
  </Action>
</Process-Node>
```

## QueryServiceInstanceAll

`com.hp.ov.activator.mwfm.component.builtin.QueryServiceInstanceAll`

According to a given service ID, the node fetches all service instance parameters from the Inventory.

**Table 4-100** QueryServiceInstanceAll Parameters

Name	Required	Description	Default	Type
<i>service_id</i>	Yes	The ID of the service instance to fetch.	None	String
<i>db</i>	No	Name of the module to access the database.	“db”	String

## QueryUCMDBCIsAndRelations

`com.hp.ov.activator.mwfm.component.builtin.QueryUCMDBCIsAndRelations`

The QueryUCMDBCIsAndRelations node query the ucldb for the specified CIs and Relations from the uCMDB. This node sets the response from ucldb in the response object.

The node throws a UCMDBException in case of an error while executing the query. The response from uCMDB is received as a UCMDBResponse object. This object is a value object, which has methods getCIs and get Relations. The getCIs method returns a List<UCMDBCI> objects and the getRelations return a List<UCMDBRelation> object. Both these objects are value objects

Both the UCMDBCI and the UCMDBRelation objects have the following properties:

- Id - id of the object as a String value
- Object type - Ci/Relation type
- stringProperties - the value of the object is a Map<String, String>
- intStrProperties- the value of the object is a Map<String, String>
- booleanProperties- the value of the object is a Map<String, Boolean>
- floatProperties- the value of the object is a Map<String, Float>
- longProperties - the value of the object is a Map<String, Long>
- doubleProperties- the value of the object is a Map<String, Double>
- dateStringProperties- the value of the object is a Map<String, String>
- xmlProperties- the value of the object is a Map<String, String>
- stringListProperties- the value of the object is a Map<String, List<String>>
- intListProperties- the value of the object is a Map<String, List<Integer>>

In addition the UCMDBReleation has the following properties

- end1Id- Id of the CI at end 1 of the relation as a String value
- end2Id - Id of the CI at end 2 of the relation as a String value

The `max_objects_returned` parameter determines the number of objects(includes both CIs and Relations) returned from by this node. The default value for this parameter is -1, which means all the results returned by uCMDB will be returned by this method. Results from uCMDB may be returned as chunks. Chunks may contain both CIs and Relations. In case a chunk contains both CIs and relations and if the number of CIs returned is more than

The node throws a UCMDBException in case there is an error while processing the request.

**Table 4-101**

### QueryUCMDBCIsAndRelations Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the UCMDBRequestModule to be used	None	String

**Table 4-101 QueryUCMDBCIsAndRelations Parameters**

Name	Required	Description	Default	Type
response	Yes	The name of the case-packet variable name in which the result is stored. The type of the case-packet variable should be Object. The result returned will be a UCMDBResponse Object. The list of UCMDBCIs can be obtained by calling a getCIs and the list of UCMDBRelations can be obtained by calling a getRelations on this object.		Object
query_name	Yes	GetCIsById GetCIsByType GetCINeighbours ExecuteTopologyQueryByName ExecuteTopologyQueryByNameWithParameters ExecuteTopologyQueryWithParameters	None	String
query_ci_type0 query_ci_type1 ... query_ci_type N	Yes(If if qualifier_p rop)	Types of CIs/Relations that need to be retrieved from uCMDB	None	String
qualifier_prop 0 qualifier_prop 1 .... qualifier_prop N	No	QualifierProperties for the query	None	String

**Table 4-101 QueryUCMDBCIsAndRelations Parameters**

Name	Required	Description	Default	Type
prop_type0 prop_type1... prop_typeN	No	The types of properties that needs to be retrieved. This can be from the following  CONCRETE- Returns the properties specific to the CI/Relation type  NAMING- Returns the naming attributes of the CI/Relation  DERIVED- Returns the attributes that have been derived from the superclass of the corresponding CI or Relation  Multiple values can be specified delimited by the # character in the form  CONCRETE#NAMING	None	String
ucmdb_object_type0 ucmdb_object_type1 ... ucmdb_object_typeN	No	The ucmdb object type for which the properties above have been specified. Valid values are ci, relation and all	None	String
query_key_name0 query_key_name1 ... query_key_name2	No	Name of the parameter being passed to the query. This can be  Id Type NeighbourType TQLName QueryXML  The id parameter can have multiple values. Hence each value needs to have a query_key_name and a corresponding query_key_value.	None	String



**Table 4-101 QueryUCMDBCIsAndRelations Parameters**

Name	Required	Description	Default	Type
max_objects_re turned	No	Maximum number of CIs/Relations that needs to be returned from the actual query results. The default value is -1 indicating all the CIs/Relations present in the query result is returned.	-1	Integer
The following properties are taken into consideration only if the query_name specified is (in this case at least one of each of the following parameters must be given): ExecuteTopologyQueryByNameWithParameters or ExecuteTopologyQueryWithParameters				
ci_id0 ci_id1... ci_idN	No	Temporary id to specify parameters.	None	String
ci_type0 ci_type1... ci_typeN	Yes (If ci_id parametes have been specified)	Label corresponding to the parametrized node in the query. This should be the same as specified	None	String
ci_prop_name0 ci_prop_name1 ... ci_prop_name N	No	Name of the property to be associated with the node in the query	None	String
ci_prop_value0 ci_prop_value1 ... ci_prop_value N	No	Value of the property name specified earlier. In case the property type is StringList or IntList then the property values can be a list of values. This can be specified by separating the values with the # character. The ci_prop_value can also be specified as a case-packet variable. In case the property type is a StringList or an IntList then the case-packet variable has to be of type Object, Internally it can contain either a String[] or a List	None	String

**Table 4-101**      **QueryUCMDBCIsAndRelations Parameters**

Name	Required	Description	Default	Type
ci_prop_type0	No	The type of the property. This can take the following values:  String Byte Integer Long Float Double Boolean Date XML StringList IntList	None	String

## RandomInteger

`com.hp.ov.activator.mwfm.component.builtin.RandomInteger`

The node allows the mwfm to generate a random number between 0 and a maximum specified number.

The node accepts *max\_number* parameter, which specifies the maximum random number to be returned. If unset, it defaults to 100. The random number is generated and stored in the case-packet mapped to the action parameter *output\_var*. This must be of Integer type.

**Table 4-102 RandomInteger Parameters**

Name	Required	Description	Default	Type
<i>max_number</i>	No	Maximum random number generated - default is 100.	100	Integer
<i>output_var</i>	Yes	Name of the case-packet variable to return the generated number.	None	Integer

**Example 4-73 Random number with default "max\_number"**

```
<Process-Node disablePersistence="true">
  <Name>RandomInteger</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.RandomInteger
    </Class-Name>
    <Param name="output_var" value="randomNum" />
  </Action>
</Process-Node>
```

**Example 4-74 Random number with "max\_number" set to 10**

```
<Process-Node disablePersistence = "true">
  <Name>RandomInteger</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.RandomInteger
    </Class-Name>
    <Param name="max_number" value="constant:10"/>
    <Param name="output_var" value="randomNum" />
  </Action>
</Process-Node>
```

## ReadData

`com.hp.ov.activator.mwfm.component.builtin.ReadData`

The node reads data saved in the `DATABASE_MESSAGE` table in the database, file system, or the string part of the `message_url`. The `message_url` can have the format `db:<message id>`, `file:<file name>`, or `data:<string>`.

This node is useful to use if the workflow does not know e.g. how the `socketlistenermodule` is configured.

If `message_url` is set to `data:<data string>` then the `<string>` is returned in the case packet defined in `output_var` parameter.

If `message_url` is set to `file:<file name>` then the `<file name>` part is used to find the file and the content of the file is returned in the case packet defined in `output_var` parameter.

If `message_url` is not specified or `message_url` is set to `db:<messageid>` the data will always be read from the database. The data can be retrieved in a number of different ways by specifying different query parameters. If no rows are found in the database `RET_VALUE` is set to 1 else 0. The data can be queried using any of the following parameters:

- `message_url`
- `job_id`
- `module_name`
- `hostname`
- `identifier`
- `identifier` and `module_name`
- `module_name` and `hostname`

The `message_url` query will return a `DatabaseMessagesDescriptor` bean. All the other queries will return an array of `DatabaseMessagesDescriptor` beans.

The retrieved data from the database is stored in the case packet mapped to the action parameter `output_value`.

The type of the `output_value` case-packet is very critical. For `message_url` based query, this parameter can either be of type `String` or `Object`. But, for all the other queries, this parameter should be `Object`. For `message_url` query, if this parameter is of type `String`, the `ReadDataFromDatabase` node will not return the `DatabaseMessagesDescriptor` bean. Instead, it will extract the message stored in the `message` field of `DATABASE_MESSAGE` table and will convert this data and will set it to the `out_value` action parameter.

Two optional parameters `data_length` and `data_position` are also provided to retrieve partial data. Partial retrieval is allowed only with `message_url` based query.

The `return_object_type` action parameter is an optional parameter. If this is configured, the data in the `message` column of the `DATABASE_MESSAGE` table will be converted to the type. The parameter should contain the fully qualified package name.

**Table 4-103 ReadData Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>message_url</i>	No	Name of the case packet variable containing the message id. The syntax is <code>db:&lt;message_id&gt;</code> , <code>file:&lt;file name&gt;</code> , or <code>data:&lt;string&gt;</code> .	None	String
<i>job_id</i>	No	Name of the case packet variable containing the job id	None	String
<i>module_name</i>	No	Name of the case packet variable containing the module name	None	String
<i>hostname</i>	No	Name of the case packet variable containing the hostname	None	String
<i>identifier</i>	No	Name of the case packet variable containing the identifier	None	String
<i>output_value</i>	Yes	When the node is executed, this case packet variable is set to data read from the database. See the description above. The value for this parameter must be a case-packet variable of type String or Object.	None	Object or String
<i>data_length</i>	No	The value of this parameter is the number of consecutive bytes of the data to be retrieved.	None	Integer
<i>data_position</i>	No	The value of parameter is the ordinal position from where the data byte has to be extracted.	None	Integer
<i>return_object_type</i>	No	This parameter indicate which java type the data in the message field should be converted to. If no parameter is specified the data will be returned as a byte array.	None	String
<i>charset</i>	No	The charset to be used when converting retrieved data to a String.	None	String

**Example 4-75 Retrieve Complete Data**

The case packet `newMessageId` contains the message id. The retrieved data would be stored in the case packet `retrievedData`.

```
<Process-Node disablePersistence="true">
  <Name>ReadDataFromDatabase</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ReadData
    </Class-Name>
    <Param name="message_url" value="newMessageId" />
    <Param name="output_value" value="retrievedData" />
  </Action>
</Process-Node>
```

#### Example 4-76 Retrieve Partial Data

The case packet messageId contains the message id. The retrieved data would be stored in the case packet retrievedData, and it contains data upto "30" consecutive bytes starting from position "1".

```
<Process-Node disablePersistence="true">
  <Name>ReadFirstString</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ReadData
    </Class-Name>
    <Param name="data_length" value="30" />
    <Param name="data_position" value="1" />
    <Param name="message_url" value="messageid" />
    <Param name="output_value" value="retrievedData" />
  </Action>
</Process-Node>
```

## ReadDataFromDatabase

`com.hp.ov.activator.mwfm.component.builtin.ReadDataFromDatabase`

The node reads data saved in the DATABASE\_MESSAGE table in the database.

The data can be retrieved in a number of different ways by specifying different query parameters. If no rows are found in the database RET\_VALUE is set to 1 else 0. The data can be queried using any of the following parameters:

- message\_url
- job\_id
- module\_name
- hostname
- identifier
- identifier and module\_name
- module\_name and hostname

The message\_url query will return a DatabaseMessagesDescriptor bean. All the other queries will return an array of DatabaseMessagesDescriptor beans.

The retrieved data from the database is stored in the case packet mapped to the action parameter *output\_value*.

The type of the output\_value case-packet is very critical. For message\_url based query, this parameter can either be of type String or Object. But, for all the other queries, this parameter should be Object. For message\_url query, if this parameter is of type String, the ReadDataFromDatabase node will not return the DatabaseMessagesDescriptor bean. Instead, it will extract the message stored in the message field of DATABASE\_MESSAGE table and will convert this data and will set it to the out\_value action parameter.

Two optional parameters data\_length and data\_position are also provided to retrieve partial data. Partial retrieval is allowed only with message\_url based query.

The return\_object\_type action parameter is an optional parameter. If this is configured, the data in the message column of the DATABASE\_MESSAGE table will be converted to the type. The parameter should contain the fully qualified package name.

**Table 4-104 ReadDataFromDatabase Parameters**

Name	Required	Description	Default	Type
<i>message_url</i>	No	Name of the case packet variable containing the message id. The syntax is db:<message_id>.	None	String
<i>job_id</i>	No	Name of the case packet variable containing the job id	None	String
<i>module_name</i>	No	Name of the case packet variable containing the module name	None	String
<i>hostname</i>	No	Name of the case packet variable containing the hostname	None	String

**Table 4-104 ReadDataFromDatabase Parameters (Continued)**

Name	Required	Description	Default	Type
<i>identifier</i>	No	Name of the case packet variable containing the identifier	None	String
<i>output_value</i>	Yes	When the node is executed, this case packet variable is set to data read from the database. See the description above. The value for this parameter must be a case-packet variable of type String or Object.	None	Object or String
<i>data_length</i>	No	The value of this parameter is the number of consecutive bytes of the data to be retrieved.	None	Integer
<i>data_position</i>	No	The value of parameter is the ordinal position from where the data byte has to be extracted.	None	Integer
<i>return_object_type</i>	No	This parameter indicate which java type the data in the message field should be converted to. If no parameter is specified the data will be returned as a byte array.	None	String
<i>charset</i>	No	The charset to be used when converting retrieved data to a String.	None	String

**Example 4-77 Retrieve Complete Data**

The case packet `newMessageId` contains the message id. The retrieved data would be stored in the case packet `retrievedData`.

```
<Process-Node disablePersistence="true">
  <Name>ReadDataFromDatabase</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ReadDataFromDatabase
    </Class-Name>
    <Param name="message_url" value="newMessageId" />
    <Param name="output_value" value="retrievedData" />
  </Action>
</Process-Node>
```

**Example 4-78 Retrieve Partial Data**

The case packet `messageId` contains the message id. The retrieved data would be stored in the case packet `retrievedData`, and it contains data upto "30" consecutive bytes starting from position "1".

```
<Process-Node disablePersistence="true">
  <Name>ReadFirstString</Name>
  <Action>
```



```
<Class-Name>  
    com.hp.ov.activator.mwfm.component.builtin.ReadDataFromDatabase  
</Class-Name>  
<Param name="data_length" value="30" />  
<Param name="data_position" value="1" />  
<Param name="message_url" value="messageid" />  
<Param name="output_value" value="retrievedData" />  
</Action>  
</Process-Node>
```

## ReadFile

`com.hp.ov.activator.mwfm.component.builtin.ReadFile`

The node reads a text file into a case-packet variable.

**Table 4-105** ReadFile Parameters

Name	Required	Description	Default	Type
<i>file</i>	Yes	Name of the file to read. The value of this parameter can be a case-packet variable that contains the name of the file, or can be a constant (specified as <code>constant:X</code> where <i>X</i> is the name of the file).  If the path name to the file is not an absolute path, the file is read relative to <code>\$ACTIVATOR_VAR</code> .	None	String
<i>destination</i>	Yes	Name of the case-packet variable to put the contents of the file into.	None	String
<i>charset</i>	No	The charset to be used when converting retrieved data to a String.	None	String

**Example 4-79** ReadFile - use in the workflow

This example illustrates how to read a file called `/tmp/example_file.txt` into a case-packet variable called `string_variable`.

```
<Process-Node disablePersistence="true">
  <Name>Read sample file</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ReadFile
    </Class-Name>
    <Param name="file" value="constant:/tmp/example_file.txt"/>
    <Param name="destination" value="string_variable"/>
  </Action>
</Process-Node>
<Case-Packet>
  <Variable name="string_variable" type="String"/>
</Case-Packet>
```

## RediscoverHostsNNMNode

`com.hp.ov.activator.mwfm.component.builtin.nnmrequest.RediscoverHostsNNMNode`

The node requests NNM to rediscover a set of network elements identified by host names or IP addresses.

Rediscover a network element means rediscover/look for changes on the network element and the interfaces. This operation is performed by the NNM on a regular basis every hour (this is configurable). If for example a new interface has been added to a network element and the user runs the rediscover operation. Then the new interface will be added to the NNM system at that moment instead of relying on the automatic rediscover period.

**Table 4-106** RediscoverHostsNNMNode Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NNMi module used to connect to a specific NNMi server	None	String
<i>host_to_rediscover0</i> , <i>host_to_rediscover1</i> , ... <i>host_to_rediscoverN</i>	Yes, at least one	The Hostname or IP address to rediscover	None	String

## ReleaseResource

`com.hp.ov.activator.mwfm.component.builtin.ReleaseResource`

The node releases poolable resources back into the inventory. To release a resource, specify a variable that contains a JavaBean that represents the resource to be released. This can be one or more case-packet variables that you list explicitly. If you do not specify any variables, the case-packet variable named `RESERVATIONS` is assumed, and all the reserved resources in that variable are released. After successful release, each released resource is removed from the `RESERVATIONS` variable.

Multiple resources can be released in a single database transaction. Successful release of all the specified resources sets the `RET_VALUE` to 0.

### See Also

- “ReserveResource” on page 281 for more information about the `RESERVATIONS` variable
- “ConfirmResourceReservation” on page 118 for more information about the `RESERVATIONS` variable

**Table 4-107 ReleaseResource Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Database module to use in order to perform the query.	“db”	String
<i>variable0, variable1... variableN</i>	No	Name of a case-packet variable that holds the resource to be released. If you do not specify any variables, all reserved resources in the <code>RESERVATIONS</code> variable are released.	None	Object

**Example 4-80 ReleaseResource - use in the workflow**

The following example releases a previously reserved UID.

```
<Process-Node>
  <Name>Release UID</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ReleaseResource
    </Class-Name>
    <Param name="variable0" value="uid"/>
  /Action>
</Process-Node>
```

## RemoveData

`com.hp.ov.activator.mwfm.component.builtin.RemoveData`

The node allows the `mwfm` to delete the data saved the `DATABASE_MESSAGE` in the database table or file system.

The data can be located either by using a message identifier or a file path, which is stored in the case-packet variable mapped to the action parameter `url_name`. You can delete the data in the database by specifying a message id, and the syntax is: `db:<message id>`. Alternatively, a file can also be deleted by specifying the file path, and the syntax is `file:<file path>`. The file path can be either absolute or relative. If a relative path is specified, the system tries to delete the file from `$ACTIVATOR_VAR` directory.

Alternatively the data can, if save in the `DATABASE_MESSAGE` table, also be located by specifying either the `job_id`, `identifier`, `module_name`, or `hostname`.

The "delete\_count" action parameter will hold the number of rows deleted.

Table 4-108

### RemoveData Parameters

Name	Required	Description	Default	Type
<code>url_name</code>	No	Name of the case-packet variable holding the message id or the file path ( an absolute path, or a filename relative to <code>\$ACTIVATOR_VAR</code> ) to be removed. The syntax is <code>db:message_id</code> or <code>file:file_path</code> .	None	String
<code>job_id</code>	No	Name of the case-packet variable holding the job id .	None	String
<code>identifier</code>	No	Name of the case-packet variable holding the identifier.	None	String
<code>module_name</code>	No	Name of the case-packet variable holding the module_name.	None	String
<code>hostname</code>	No	Name of the case-packet variable holding the hostname.	None	String
<code>delete_count</code>	No	This case packet variable, if defined, will contain the count of rows deleted by the node.	None	Integer

Example 4-81

### Remove data from database

```
<Process-Node>
  <Name>RemoveData</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.RemoveData
    </Class-Name>
    <Param name="url_name" value="messageId"/>
  </Action>
</Process-Node>
```

The case-packet messageID must contain the message id, and its value must be db:<message id>.

**Example 4-82 Remove data from file system, absolute path**

```
<Process-Node>
  <Name>RemoveData</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.RemoveData
    </Class-Name>
    <Param name="url_name" value="constant:file:c:/hp/ActivationData.txt"/>
  </Action>
</Process-Node>
```

**Example 4-83 Remove data from file system, relative path**

```
<Process-Node>
  <Name>RemoveData</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.RemoveData
    </Class-Name>
    <Param name="url_name" value="constant:file:ActivationData.txt"/>
  </Action>
</Process-Node>
```

The system tries to locate the file and delete it from \$ACTIVATOR\_VAR directory.

## RemoveFile

`com.hp.ov.activator.mwfm.component.builtin.RemoveFile`

The node deletes a file.

**Table 4-109** RemoveFile Parameters

Name	Required	Description	Default	Type
<i>file</i>	Yes	Name of the file to be removed. The value of this parameter can be a case-packet variable that contains the name of the file, or can be a constant (specified as <code>constant:X</code> where <i>X</i> is the name of the file).  If the path name to the file is not an absolute path, the file is read relative to <code>\$ACTIVATOR_VAR</code> .	None	String

## Replace

`com.hp.ov.activator.mwfm.component.builtin.Replace`

The node carries out simple character substitutions.

**Table 4-110** Replace Parameters

Name	Required	Description	Default	Type
<i>variable</i>	Yes	Name of the case-packet variable in which substitutions are performed.	None	String
<i>origin</i>	Yes	Character to be replaced. You can specify <code>\n</code> to indicate a carriage return.	None	String
<i>destination</i>	Yes	Destination character. You can specify <code>\n</code> .	None	String

**Example 4-84** Replace - use in the workflow

This example replaces all instances of the letter “a” with the letter “A” in the string case-packet variable.

```
<Process-Node disablePersistence="true">
  <Name>Replace a's</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Replace
    </Class-Name>
    <Param name="variable" value="string"/>
    <Param name="origin" value="a"/>
    <Param name="destination" value="A"/>
  </Action>
</Process-Node>
```



## ReserveResource

`com.hp.ov.activator.mwfm.component.builtin.ReserveResource`

The node is used to reserve resources from the inventory.

Multiple resources can be reserved in a single transaction. If any of the specified JavaBeans does not have a resource available for reservation, the transaction is rolled back and the `RET_VALUE` variable is set to 1 to indicate a failure. Successful reservation of all the specified resources sets the `RET_VALUE` to 0. In cases where the number of resources to be reserved is not known before runtime, it is possible to use the workflow `AppendToResourceList` workflow node to dynamically build a list of resources to be reserved in a single database transaction; in this case, use the `resource_list_var` parameter to pass the list of resources to the `ReserveResource` node.

The reservation saves a reference to each JavaBean in a specified case-packet variable. In the workflow, you can refer to an individual field in the JavaBean using a special syntax. The syntax is similar to referencing a member of the Java class `variable.field`. The field name depends on the definition of the JavaBean that was specified when `InventoryBuilder` was used. See the example below for how this syntax is used.

It is also possible to use this node to reserve a resource by a composed foreign key; however, in this case it is only possible to reserve a single resource. To reserve a resource by a composed foreign key only a single bean and variable parameter can be specified while multiple `key_field` and `key_value` parameters may be specified.

If a case-packet variable with the name `RESERVATIONS` exists in the case-packet, then all of the reserved resources are added to it. The `RESERVATIONS` case-packet variable can be used to ensure that all reserved resources are released (for instance, in case of unexpected errors)

### See Also

- “ReleaseResource” on page 276 for more information about the `RESERVATIONS` variable
- “AppendToResourceList” on page 101
- “ConfirmResourceReservation” on page 118 for more information about the `RESERVATIONS` variable

**Table 4-111 ReserveResource Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Database module to use in order to perform the query.	“db”	String

**Table 4-111 ReserveResource Parameters (Continued)**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>bean0,</i> <i>bean1...</i> <i>beanN</i>	No	Name of the Java class generated by Inventory Builder for a reservable resource from which to allocate the resource.  The use of this parameter is mandatory if resource_list_var is not specified.	None	String
<i>key_field0,</i> <i>key_field1...</i> <i>key_fieldN</i>	No	Name of the field by which the reservation is carried out. This defaults to doing a reservation by PrimaryKey.	None	String
<i>key_value0,</i> <i>key_value1...</i> <i>key_valueN</i>	No	Name of a case-packet variable that contains the value of the key that is to be used for the reservation.	None	Depends on the bean
<i>variable0,</i> <i>variable1...</i> <i>variableN</i>	Yes	Name of a case-packet variable in which to catch the reserved resource.	None	Object
<i>resource_list_var</i>	No	Specifying the list containing information about which resources should be reserved  If this parameter is specified, it is not possible to specify any beanN, key_fieldN, and key_valueN parameters.	None	Object
<i>output_list_var</i>	No	Name of a case packet variable to catch the reserved resource list. Mandatory if resource_list_var is present.  If this parameter is specified, it is not possible to specify any variableN parameters.	None	Object

**Example 4-85 ReserveResource - use in the workflow**

This example reserves the next available UID, and then prints a message that contains the reserved ID.

```
<Process-Node>
  <Name>Reserve UID</Name>
  <Description>Reserves a free UID in the web server.</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ReserveResource
    </Class-Name>
    <Param name="bean0" value="com.hp.ov.activator.example.UID"/>
    <Param name="key_field0" value="WebServer"/>
    <Param name="key_value0" value="web_server_name"/>
    <Param name="variable0" value="uid"/>
  </Action>
</Process-Node>

<Process-Node>
  <Name>Tell operator</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.PutMessage
    </Class-Name>
    <Param name="queue" value="reservationInfo"/>
    <Param name="message" value="Reserved UID %s in WebServer %s"/>
    <Param name="param1" value="uid.web_server"/>
    <Param name="param0" value="uid.userid"/>
  </Action>
</Process-Node>
```

**Example 4-86 ReserveResource - use in the workflow**

This example reserves the next available IP number within a given region and city..

```
<Process-Node>
  <Name>ReserveResource</Name>
  <Description>Reserves a free IP number in a given region/city.</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.ReserveResource
    </Class-Name>
    <Param name="bean0" value="com.hp.ov.activator.example.IPNumber"/>
    <Param name="key_field0" value="region"/>
    <Param name="key_field1" value="city"/>
    <Param name="key_value0" value="region_variable"/>
    <Param name="key_value1" value="city_variable"/>
    <Param name="variable0" value="reserved_ip_number"/>
  </Action>
</Process-Node>
```

## RetrieveSequence

`com.hp.ov.activator.mwfm.component.builtin.RetrieveSequence`

The node generates a unique sequence number based on an Database sequence object. It is up to the workflow designer to ensure the sequence number exists in the database

**Table 4-112 RetrieveSequence Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	The db module used to connect to the database. If unspecified in the workflow, it defaults to 'db'.	"db"	String
<i>sequence_name</i>	Yes	Name of the specific sequence whose next value has to be retrieved from the database.	None	String
<i>sequence_value</i>	No	Holds the next value of the sequence as retrieved from the database by the process node.	None	Object

**Example 4-87 RetrieveSequence**

The node retrieves the next value from the specified database sequence name in *sequence\_name* attribute and stores it in the case-packed variable assigned to *sequence\_value*.

The reason for having the output parameter of type object is it then makes it possible to return either the value as string or integer depending on the case-packet type.

```
<Process-Node>
  <Name>RetrieveSequence</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.RetrieveSequence
    </Class-Name>
    <Param name="sequence_name" value="constant:TEST_WORKFLOW_NODE" />
    <Param name="sequence_value" value="seq_next_value" />
  </Action>
</Process-Node>
```

## ScheduleCurrentJob

`com.hp.ov.activator.mwfm.component.builtin.ScheduleCurrentJob`

This node allows you to schedule a running job for later execution. This means that the job stops executing and gets persisted completely into the database. The job does not consume any resources while stored in the database.

Use the *sleep\_time* parameter to suspend the running job for a certain period of time or use the *schedule\_time* parameter to suspend the job until a specified date and time.

To schedule a running job, you must specify the ID of the running job and either *sleep\_time* or *schedule\_time*.

If the node finishes without errors, the RET\_VALUE case-packet variable is set to 0. In case of any error in the node, RET\_VALUE is set to 1 and the RET\_TEXT case-packet variable holds more information about the problem. If the job being scheduled is not on the list of scheduled jobs, the node sets RET\_VALUE to 1, puts an error description in RET\_TEXT and continues to the next node. It is important to remember that the job being scheduled cannot have reoccurrence.

**Table 4-113** ScheduleCurrentJob Parameters

Name	Required	Description	Default	Type
<i>schedule_time</i>	No	The date and the time when the running job must resume. This parameter accepts date and time as milliseconds with the value starting from January 1, 1970 00:0000:000 GMT. The value should be numeric.	0	Integer
<i>sleep_time</i>	No	Sleep time to supend the running job for a certain period of time. The sleep time must be specified in seconds.	0	Integer
<i>group</i>	No	The group id of scheduled job.	0	String
<i>description</i>	No	The description of scheduled job.	0	String
<i>status</i>	No	The status of the scheduled job	0	String

## ScheduleJob

`com.hp.ov.activator.mwfm.component.builtin.ScheduleJob`

The node schedules a workflow for a specified time. The node works similarly to the `StartJob` node. It checks if the workflow to schedule has no errors, and only then adds it to the list of scheduled jobs. You can specify the schedule time, group id, status and description of a scheduled job. If you want to run the scheduled job repeatedly, specify repeating frequency and repeating end time.

If the node finishes without errors, the `RET_VALUE` case-packet variable is set to 0. In case of any error in the node, `RET_VALUE` is set to 1. The `RET_TEXT` case-packet variable holds more information about the problem.

### See Also

- “StartJob” on page 295 for more information.

**Table 4-114** ScheduleJob Parameters

Name	Required	Description	Default	Type
<i>workflow_name</i>	Yes	The name of the workflow to schedule	None	String
<i>schedule_time</i>	Yes	The date and time to start the workflow. This parameter accepts the date and time as milliseconds with the starting value from January 1, 1970 00:00.00:000 GMT. The value should be numeric.	None	Integer
<i>group_id</i>	No	This parameter is used to group a set of scheduled jobs. It is useful in connection with timed services and reoccurring scheduled workflows where a common identifier is needed.	None	String
<i>reoccurrence_freq</i>	No	Reoccurrence frequency period. This parameter has to be specified if the scheduled job has to be run repeatedly. If <code>reoccurrence_freq</code> is not specified, the parameter accepts the value as seconds. The value should be numeric.	0	Integer

**Table 4-114 ScheduleJob Parameters (Continued)**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>reoccurrence_end_time</i>	No	The time when reoccurrence of schedule job must end. This parameter accepts the date and time as milliseconds with the value starting value from January 1, 1970 00:00.00:000 GMT. The value should be numeric.	0	Integer
<i>description</i>	No	Description of the scheduled job. The default value is an empty string.	None	String
<i>status</i>	No	Status of the scheduled job. The default value is an empty string.	None	String
<i>variable0, variable1, ... VariableN</i>	No	Case-packet variables from the current workflow that must be passed to scheduled workflow. This parameter must always be a case-packet variable. You can specify any number of variable{x} starting from variable0.	None	Any
<i>destination0, destination1, .. destinationN</i>	No	Name of the case-packet variable, in the workflow being started, that must be initialized with the value of the case-packet variable specified by variable{x} of the same number. Per default the name of the case-packet variable to be initialize in the child workflow is assumed be the same as the parents.	None	Depends on the variable
<i>ret_scheduled_job_id</i>	No	If the scheduled job was successfully started the Job Id of the newly scheduled job is returned into this case-packet variable. On error the case-packet variable specified will get the value -1.	-1	Integer

**Table 4-114**      **ScheduleJob Parameters (Continued)**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>reoccurrence_freq_units</i>	No	Reoccurrence frequency units: 1-second, 2 minutes, 3-hours, 4-days, 5-weeks, 6-months. The default value is 1-second. The value should be numeric and can start from 1 to 6.	1	Integer



## SendAlarm

`com.hp.ov.activator.mwfm.component.builtin.SendAlarm`

The node is used to send messages to any workflow module that implements the MessageModule interface. This is typically used for sending a message to OVO using the OVOMessageModule.

### See Also

- “OVOMessageModule” on page 415 for more information about how to send messages to OVO.

**Table 4-115** SendAlarm Parameters

Name	Required	Description	Default	Type
<i>module</i>	Yes	Name of a module that will take care of sending the message.	None	String
<i>message</i>	Yes	Message to be sent. By default, the message is specified as a case-packet variable that contains the message to be sent. To indicate a hard-coded message, use the syntax <code>constant :X</code> . The message can contain free variables (indicated by <code>%s</code> symbol) to be replaced with other parameters.	None	String
<i>param0, param1... paramN</i>	No	Names of case-packet variables whose values replace the free variables in the message. Specify as many <code>param</code> as needed.	None	Any

The SendAlarm node has a special way of handling parameters. It allows you to pass additional parameters to the MessageModule that the SendAlarm node may be unaware of. For example, the OVOMessageModule supports a special parameter to set the severity of the message. The special parameters supported by the OVOMessageModule are listed below. Note that these parameters may be supported by another module that you may use instead of the OVOMessageModule.

**Table 4-116 SendAlarm Parameters - for use by OVOMessageModule**

Name	Required	Description	Default	Type
<i>severity</i>	No	Set the severity of the message that is being sent to OVO. This overrides the default severity set in the OVOMessageModule configuration.	None	
<i>application</i>	No	Set the indicator of the name of the application from which the message is coming. This overrides the default application name, which is typically "ServiceActivator".	None	
<i>object</i>	No	Set the name of the object on whose behalf the message will be sent. This overrides the default (blank).	None	
<i>msg_grp</i>	No	Set the message group of the message that is being sent to OVO. This overrides the default message group set in the configuration of the OVOMessageModule.	None	
<i>node</i>	No	Set the name of the node on whose behalf the message will be sent. This overrides the default (null).	None	
<i>service_id</i>	No	Set the service_id for the message that is being sent to OVO. This overrides the default (blank).	None	

**Example 4-88 SendAlarm - use in the workflow**

This is an example of an alarm for the OVO MessageModule that uses two custom parameters called *severity* and *object*.

```
<Process-Node>
  <Name>Send Alarm</Name>
  <Description>Send an alarm to OVO</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.SendAlarm
    </Class-Name>
    <Param name="module" value="ovo" />
    <Param name="message" value="Failure to activate the %s service." />
    <Param name="param0" value="service_name" />
    <Param name="severity" value="constant:critical" />
    <Param name="object" value="machine" />
  </Action>
</Process-Node>
```

## SendMessage

`com.hp.ov.activator.mwfm.component.builtin.SendMessage`

The node sends messages using a SenderModule. It can use any module that implements the SenderModule interface. The typical module used in this case is the SocketSenderModule.

A message to be sent can come from a case-packet variable, or from a file, or from the database.

When the node completes, the value of the built-in case-packet variable `RET_VALUE` is set to 0 if the message was properly enqueued and to 1 if not.

### See Also

- “SocketSenderModule” on page 432 for more information about how to send messages to a waiting program.

**Table 4-117** SendMessage Parameters

Name	Required	Description	Default	Type
<i>sender</i>	Yes	Name of the Workflow Manager module that will send this message.	None	String
<i>message_var</i>	Yes, if <i>message_url</i> is not used	Name of a case-packet variable that contains the message to be sent.	None	String
<i>message_url</i>	Yes, if <i>message_var</i> is not used	Name of a case-packet variable that contains the name of the file or a message id representing a row in <code>DATABASE_MESSAGE</code> containing the message. The syntax is <code>db:message_id</code> or <code>file:file_path</code> . A constant file name or message id can also be specified.	None	String

### Example 4-89 SendMessage - use in the workflow

This example shows the `SendMessage` node being used to send a message via the `tcp_example_sender` module. The message to be sent is in the case-packet variable `returnMessage`, and the value is `file:<file path>`.

```
<Process-Node>
  <Name>Send Message</Name>
  <Description>Send a message to the CRM</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.SendMessage
    </Class-Name>
    <Param name="sender" value="tcp_example_sender"/>
    <Param name="message_url" value="returnMessage"/>
  </Action>
</Process-Node>
```

**Example 4-90      SendMessage - using a constant message\_id parameter**

```
<Process-Node>
  <Name>Send Message</Name>
  <Description>Send a message to the CRM</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.SendMessage
    </Class-Name>
    <Param name="sender" value="tcp_example_sender"/>
    <Param name="message_url" value="db:1"/>
  </Action>
</Process-Node>
```

**Example 4-91      SendMessage - using a variable message\_id parameter**

```
<Process-Node>
  <Name>Send Message</Name>
  <Description>Send a message to the CRM</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.SendMessage
    </Class-Name>
    <Param name="sender" value="tcp_example_sender"/>
    <Param name="message_url" value="messageId"/>
  </Action>
</Process-Node>
</Case-Packet>
<Variable name="messageId" type="Integer"/>
<Initial-Case-Packet>
  <Variable-value name="messageId" value="db:2"/>
</Initial-Case-Packet>
```

## SendSNMPTrap

`com.hp.ov.activator.mwfm.component.builtin.SendSNMPTrap`

This node will enable a workflow to send a custom SNMP trap using the `SNMP Sender Module`. The `SNMP Sender Module` should have been configured for this node to function. This will enable solutions to send custom traps.

The parameter `trap_oid` is the oid of the trap itself. Where the `oid` and `trap_message` can be used to add additional information.

**Table 4-118 SendSNMPTrap Parameters**

Name	Required	Description	Default	Type
<i>module</i>	Yes	The SNMP Sender module to be used	None	String
<i>trap_oid</i>	Yes	Oid of the trap	None	String
<i>oid0</i> , <i>oid1</i> , ... <i>oidN</i>	No	Additional oid to include in the trap	None	String
<i>trap_message</i> , <i>trap_message1</i> , ... <i>trap_messageN</i>	No	The trap message associated with the oid	None	String

**Example 4-92 SendSNMPTrap - use in the workflow**

```
<Process-Node>
  <Name>SendSNMPTrap</Name>
  <Description></Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.SendSNMPTrap
    </Class-Name>
    <Param name="module" value="constant:snmp_sender"/>
    <Param name="trap_oid" value="constant:1.3.6.1.4.1.11.4.1"/>
    <Param name="oid0" value="constant:1.3.6.1.4.1.11.4.1"/>
    <Param name="trap_message0" value="constant:Test trap from snmp node"/>
  </Action>
</Process-Node>
```

## Sleep

`com.hp.ov.activator.mwfm.component.builtin.Sleep`

The node pauses a workflow for a specified amount of time.

**Table 4-119 Sleep Parameters**

Name	Required	Description	Default	Type
<i>time</i>	Yes	Indicates the name of a case-packet variable that contains the time to sleep (in milliseconds). A hard-coded value can be specified using the syntax <code>constant:X</code> .	None	Numeric
swap	No	Instructs the Workflow manager to swap-out the case-packets while the job waits in the request queue, in order to reduce memory footprint	false	Boolean
skip	No	if set to true then the workflow node will be skipped	false	Boolean

**Example 4-93 Sleep - use in the workflow**

This example blocks the flow for a second.

```
<Process-Node disablePersistence="true">
  <Name>Sleep node</Name>
  <Description>Blocks a workflow without using up the CPU</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Sleep
    </Class-Name>
    <Param name="time" value="constant:1000"/>
  </Action>
  <Next-Node>Another node</Next-Node>
</Process-Node>
```

## StartJob

`com.hp.ov.activator.mwfm.component.builtin.StartJob`

The node starts a new job and optionally passes some initial values to its case-packet variables.

The current workflow does not wait for the newly started workflow to complete. The current workflow proceeds directly to the next node.

### See Also

- “AskFor” on page 104 for more information about how one workflow can wait for another

**Table 4-120 StartJob Parameters**

Name	Required	Description	Default	Type
<i>workflow_name</i>	Yes	Name of a case-packet variable that contains the name of the workflow to be started. To specify a hard-coded value, use the syntax <code>constant:X</code> .	None	String
<i>variable0</i> , <i>variable1...</i> <i>variableN</i>	No	Names of case-packet variables to be passed to the child workflow.	None	Object
<i>destination0</i> , <i>destination1...</i> <i>destinationN</i>	No	Names of case-packet variables in the child workflow to receive the matching variable from this workflow. By default, the variables are passed to variables of the same name in the child workflow. Destination parameters can be specified selectively for some or each of the indicated variables.	None	Object
<i>output_job_id_variable</i>	No	A case-packet variable to catch the <code>JOB_ID</code> of the newly started job	None	Integer

#### Example 4-94 StartJob - use in the workflow

This example starts a child workflow and passes the current JOB\_ID to the child. Note that it specifies the destination variable that will receive this JOB\_ID. This is the standard way to start a child so that it can successfully communicate back to the parent.

Typically, the next node in this workflow would do an AskFor to wait for some information back from the child.

```
<Process-Node>
  <Name>Start work</Name>
  <Description>Creates another workflow</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.StartJob
    </Class-Name>
    <Param name="workflow_name" value="workflow"/>
    <Param name="variable0" value="JOB_ID" />
    <Param name="destination0" value="controller_job_id" />
    <Param name="variable1" value="message_file" />
  </Action>
</Process-Node>
```



## StartJobAndWait

`com.hp.ov.activator.mwfm.component.builtin.StartJobAndWait`

This node starts a new job and blocks until the newly started job has synchronized with its parent where after it proceeds to the next node. It is optional to pass on case-packet variables to the new job; however, the child job needs at least information about the parent's job\_id in order to synchronize with its parent. The synchronization from the child job can be done either by using the Sync node or the SyncHandler.

### See Also

- “StartJob” on page 295

**Table 4-121 StartJobAndWait Parameters**

Name	Required	Description	Default	Type
<i>workflow_name</i>	Yes	Name of the workflow to start.	None	String
<i>variable0,</i> <i>variable1...</i> <i>variableN</i>	No	Case-packet variables that are to be passed to initialize variables in the new workflow being started.	None	Any
<i>destination0,</i> <i>destination1...</i> <i>destinationN</i>	No	The name of the case-packet variable to initialize in the new workflow. By default the variable of the same name is initialized.	None	Any
<i>outputvar0,</i> <i>outputvar1...</i> <i>outputvarN</i>	No	Case-packet variables that the child should pass back.	None	Object
<i>queue</i>	No	The name of the queue where the job will wait and where the child job must do the synchronization.	sync	String
swap	No	Instructs the Workflow manager to swap-out the case-packets while the job waits in the request queue, in order to reduce memory footprint	false	Boolean

### Example 4-95 StartJobAndWait - use in the workflow

The StartJobAndWait node starts the job called “ThisJobWillBeStarted”. It sends the content of the case-packet variable called “parentVariable” to the child’s case-packet variable “childVariable”. It also sends the parent job\_id to the child case-packet variable called “sync\_jobid” from where it is used to sync with the parent later on. The queue to handle the synchronization between the jobs is set to be “JobSyncQueue”. The parent case-packet variable “passedFromChildToParent” must be set from the child before the synchronization can be accepted. The child SyncHandler uses the passed information to synchronize with the parent at the end of the child workflow.

StartJobAndWait node in the parent workflow.

```
<Name>StartJobAndWait</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.StartJobAndWait
    </Class-Name>
    <Param name="workflow_name" value="constant:ThisJobWillBeStarted"/>
    <Param name="destination0" value="childVariable"/>
    <Param name="outputvar0" value="passedFromChildToParent"/>
    <Param name="variable0" value="parentVariable"/>
    <Param name="destination1" value="sync_jobid"/>
    <Param name="variable1" value="JOB_ID"/>
    <Param name="queue" value="JobSyncQueue"/>
  </Action>
```

Child workflow synchronization using SyncHandler

```
<End-Handler>
  <Class-Name>
    com.hp.ov.activator.mwfm.component.builtin.SyncHandler
  </Class-Name>
  <Param name="job_id" value="sync_jobid"/>
  <Param name="queue" value="constant:JobSyncQueue"/>
  <Param name="destination0" value="passedFromChildToParent"/>
  <Param name="variable0" value="SendThisToTheParent"/>
</End-Handler>
```

## Switch

`com.hp.ov.activator.mwfm.component.builtin.SwitchCase`

This node allows the Workflow Manager to provide branching depending on the value of the parameter `key`. The `key` can be a constant or a case-packet variable of type `String` or `Integer`.

The case values that govern the multiple branches from the Switch node are specified using the action parameters `case0`, `case1...caseN`. In the Workflow Designer, when the Switch node is connected to another node, the user is prompted to enter the `case` value that governs this branch; this can be a constant or a case-packet variable of type `Integer` or `String`. The case parameters are displayed in a drop-down list in the "Arrow drawing window" along with the default option. The user can select either a case parameter or the default option.

The default path for the Switch node is mandatory. The case parameters are optional.

During workflow execution, when the Switch node is processed, the `key` is evaluated and an attempt is made to find the matching case value. If a match is found then the workflow node for the matching case branch becomes the next node to be processed by the workflow manager. If a match is not found the workflow node in the default branch is chosen.

**Table 4-122** Switch Parameters

Name	Required	Description	Default	Type
<i>key</i>	Yes	The key that is evaluated and which decides the workflow path to taken	None	String or Integer
<i>case0</i> <i>case1...</i> <i>caseN</i>	Yes	Case values specified for various branches.	None	Depend on the bean

**Example 4-96** Switch - use in the workflow

This example show how the workflow branch depending on the case-packet `day`.

```
<Switch-Node>
  <Name>Branch on day</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.SwitchCase
    </Class-Name>
    <Param name="key" value="day" />
    <Param name="case0" value="constant:day1" />
    <Param name="case1" value="constant:day2" />
    <Param name="case2" value="constant:day3" />
  </Action>
  <Switch name="case0">Sunday</Switch>
  <Switch name="case1">Monday</Switch>
  <Switch name="case2">Tuesday</Switch>
  <Default>DoNothing</Default>
</Switch-Node>
```

## Sync

`com.hp.ov.activator.mwfm.component.builtin.Sync`

The node responds to a workflow node that is waiting for interaction on a request queue (one that has done an `AskFor`). Use this node to synchronize a child workflow with its parent workflow.

The `Sync` node provides a way to determine whether the node actually was able to synchronize with the indicated `job_id`. The `Sync` node accepts a parameter with the name “ok.” This variable receives the value “true” if the node is able to respond to a waiting request for the given `job_id` in the given queue. The variable receives the value “false” if the given `job_id` is not waiting in the given queue. If this parameter is not used, there is no way to know if the synchronization was successful.

### See Also

- “AskFor” on page 104 for information about how one workflow can wait for input from another workflow

**Table 4-123** Sync Parameters

Name	Required	Description	Default	Type
<i>job_id</i>	Yes	Name of a case-packet variable that contains the job ID of the workflow waiting to synchronize.	None	Integer
<i>queue</i>	Yes	Name of the queue on which the workflow is waiting, specified as a constant string.	None	Queue
<i>OK</i>	No	Name of the Boolean case-packet variable to catch the indication of whether the Sync was successful or not.	None	Boolean
<i>variable0</i> , <i>variable1</i> ... <i>variableN</i>	Yes	Names of the case-packet variables to be passed to the waiting workflow. By default, the variables are passed to variables of the same name in the waiting workflow. <i>destinationN</i> parameters can be specified selectively for some or all of the indicated variables.	None	Any
<i>destination0</i> , <i>destination1</i> ... <i>destinationN</i>	No	Names of the case-packet variables in the parent workflow that are waiting to receive the matching variable from this workflow.	None	Any

**Example 4-97 Sync - use in the workflow**

This example is a child workflow attempting to synchronize with its controller workflow.

```
<Process-Node>
  <Name>Sync with controller</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.Sync
    </Class-Name>
    <Param name="job_id" value="controller_job_id" />
    <Param name="queue" value="controller_queue" />
    <Param name="variable0" value="activation_major_code" />
    <Param name="destination0" value="operation_status" />
  </Action>
</Process-Node>
```

## ThrowError

`com.hp.ov.activator.mwfm.component.builtin.ThrowError`

The node throws an error given by the message argument.

**Table 4-124**      **ThrowError Pameters**

Name	Required	Description	Default	Type
<i>message</i>	No	Message to be included as part of the error.	None	String

## ThrowException

`com.hp.ov.activator.mwfm.component.builtin.ThrowException`

The node throws an exception with the message given by the message argument.

**Table 4-125** **ThrowException Parameters**

Name	Required	Description	Default	Type
<i>message</i>	No	Message to be included as part of the exception.	None	String

## ThrowRuntimeException

`com.hp.ov.activator.mwfm.component.builtin.ThrowRuntimeException`

The node throws a runtime exception with the message given by the message argument.

**Table 4-126** **ThrowRuntimeException Parameters**

Name	Required	Description	Default	Type
<i>message</i>	No	Message to be included as part of the exception.	None	String



## TransformXML

`com.hp.ov.activator.mwfm.component.builtin.TransformXML`

The node performs an XSL transform on an XML document using standard XSLT components. You may use any XSL specification to transform any XML document.

In addition to the ability to perform an XSL transform, it is also possible to replace elements in the XSL template with the current value of case-packet variables. When using this functionality, it is not even necessary to provide an input XML document since all the necessary information may be in the XSL itself.

---

**NOTE**

There are two different syntaxes for replacing parameters in the XSL document with case-packet variables. Refer to the example below for both syntaxes.

The XML input (if needed) may come from a case-packet variable or from a URL.

The XSL input may come from a case-packet variable or from a URL.

The output may be put into a case-packet variable or maybe sent to a URL.

**Table 4-127 TransformXML Parameters**

Name	Required	Description	Default	Type
<i>xml_url</i>	no	The location of the XML input document. The location may be specified in any valid URL (http:/..., file:..., etc).  The parameter may indicate that the URL is found in a case-packet variable or as a constant in the form - constant:<url>	None	String
<i>xml_var</i>	no	The name of a case-packet variable from which to get the XML input document.	None	String
<i>xsl_url</i>	either <i>xsl_url</i> or <i>xsl_var</i>	The location of the XSL template. The location may be specified in any valid URL (http:/..., file:..., etc).  The parameter may indicate that the URL is found in a case-packet variable or as a constant in the form - constant:<url>	None	String
<i>xsl_var</i>	either <i>xsl_url</i> or <i>xsl_var</i>	The name of a case-packet variable from which to get the XSL specification.	None	String

**Table 4-127 TransformXML Parameters (Continued)**

Name	Required	Description	Default	Type
<i>output_url</i>	either output_url or output_var	The location for the output from the transform. The location may be specified in any valid URL (http:/..., file:..., etc).  The parameter may indicate that the URL is found in a case-packet variable or as a constant in the form - constant:<url>	None	String
<i>output_var</i>	either output_url or output_var	The name of a case-packet variable to catch the result of the XSL transform.	None	String
use_solution_dir	No	When set to "true", the nodes will read from \$SOLUTION_ETC/template_files instead of \$ACTIVATOR_ETC/template_files.	false	Boolean
end_of_line_style	No	The end of line style of the output document. Possible values are 'windows' or 'unix'. If not present the end of line will be the system default one	the hpsa operation system	String

**Example 4-98 TransformXML - use in the workflow**

This example creates a connection template for use in the GenericCLI plug-in. Notice that it does not provide an input XML document. The XSL template contains everything necessary to produce the desired output, it probably refers to case-packet variables, see the next example for an XSL template that would be meaningful in this case.

```

<Process-Node disablePersistence="true">
  <Name>Prepare for PIX connection</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.engine.component.builtin.TransformXML
    </Class-Name>
    <Param name="xsl_url" value="constant:file:///C:/HP/Openview/
      ServiceActivator/etc/cisco/
      CiscoPIX_telnet_direct.xsl" />
    <Param name="output_var" value="ciscoConnectString"/>
  </Action>
</Process-Node>

```

**Example 4-99 TransformXML - XSL template**

This example shows the form of an XSL template that replaces elements in the template with the value of case-packet variables. Notice the use of the `xsl:param` declaration near the top. This indicates that the template will refer to three variables in the body below. The workflow node will create XSL variables for each case-packet variable in the workflow. These can then be referred to in the XSL specification, as it is done for `pix_pswd`, `pix_enable_pswd`, and `pix_timeout`.

Notice the difference in the syntax when referring to the `pix_timeout` variable vs. the syntax used for `pix_pswd`. The different syntax is necessary because of the strict nature of XML. One syntax is necessary when referring to a parameter inside of a tag attribute. The other syntax is used to refer to the parameter as part of the element text.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:output doctype-system="CLIv4.dtd" />

<xsl:param name="pix_pswd" />
<xsl:param name="pix_enable_pswd" />
<xsl:param name="pix_timeout" />

<xsl:template match="/">

<CLI>
  <Connect protocol="telnet">
    <Do timeout="{pix_timeout}" description="PIX device authentication failed.">
      <Confirm>
        <Pattern>^PIX passwd: $</Pattern>
        <Command><xsl:value-of select="$pix_pswd"/></Command>
      </Confirm>
      <Error>^PIX passwd: $</Error>
      <Prompt>> $</Prompt>
    </Do>

    <Do description="PIX privileged (enable) mode authentication failed.">
      <Command>enable</Command>
      <Confirm>
        <Pattern>Password: $</Pattern>
        <Command><xsl:value-of select="$pix_enable_pswd"/></Command>
      </Confirm>
      <Error>Password: $</Error>
      <Error>^usage:</Error>
      <Prompt># $</Prompt>
    </Do>
  </Connect>
</CLI>
</xsl:template>
</xsl:stylesheet>
```

**Example 4-100 TransformXML - XSL template with complex data types**

This example shows the form of an XSL template that replaces elements in the template with case-packet variables of the type object. The object can be of type maps, java beans, and arrays. The syntax for specifying complex data types is the same as which is defined in *References to Complex Data Types in Workflow Node Parameters* with the modification that the special characters must be substituted as shown in Table 4-128.

**Table 4-128 Complex data type usage in xslt**

Symbol	Replacement in xslt
{	<u>LC</u>
}	<u>RC</u>
[	<u>LB</u>
]	<u>RB</u>
“	<u>Q</u>
#	<u>H</u>

Notice the use of the `xsl:param` declaration near the top. This indicates that the template will refer to variables in the body as below:

- The elements `beanCP.attribute1` is directly referring to the member variables of the Java bean object.
- The element `arrayCP [0]` is written as `arrayCPLB0RB`, where the character “[” is replaced by `LB` and the character “]” is replaced by `RB`.
- The element `hashMapcp {"key1"}` is written as `hashMapcpLCQkey1QRC`, where the character “{” is replaced by `LC` and the character “}” is replaced by `RC` and the “” is replaced by `Q`.

The other complex data types used below are combinations of maps, arrays, and bean objects.

The workflow node will create XSL variables for each case-packet variable in the workflow. These can then be referred to in the XSL specification, as it is done for `$hashMapcpLCQkey1QRC`, `$arrayCPLB0RB` and `$beanCPDattribute1`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- We want to produce output for a HTML browser -->
<xsl:output method="html"/>
<xsl:preserve-space elements="*" />
<xsl:param name="beanCP.attribute1"/>
<xsl:param name="arrayCPLB0RB" />
<xsl:param name="beanCPDattribute2LB0RB" />
<xsl:param name="hashMapcpLCQkey1QRC" />
<xsl:param name="hashMapcpLCbeanCPDattribute2LB0RBRC" />
<xsl:param
name="hashMapComplexcpLCbeanCPDattribute2LB0RBRCLCarrayCPLB1RBRCDattribute2H" />
```

```

<xsl:template match="/">
<html>
<head>
<script>
</script>
</head>
<style>
.row0 {background-color: #E6E6E6;
font-family: Verdana, Helvetica, Arial, Sans-serif;
font-size: 8pt; }
.row1 {background-color: #CCCCCC;
font-family: Verdana, Helvetica, Arial, Sans-serif;
font-size: 8pt; }
.heading {
background: #336699;
font-family: Verdana, Helvetica, Arial, Sans-serif;
font-size: 8pt;
color: white;
text-align: left;
vertical-align: middle;
border: 1px solid white;
}
</style>
<body style="font-family:Verdana, Helvetica, Arial,
Sans-serif;font-size:8pt" onmousemove="savePosition();">
<table width="100%">
<tr>
<td class="heading">Accessing attributes of Java Bean -
beanCP.attribute1</td>
<td class="heading">Accessing elements of an array - arrayCP[0]</td>
<td class="heading">Accessing elements of an array returned as attributes of
Java Bean - beanCP.attribute2[0]</td>
<td class="heading">Accessing Value of a hashMap using constant key -
hashMapcp{"key1"}</td>
<td class="heading">Accessing Value of a hashMap using elements of an array
returned as attributes of Java Bean as key - hashMapcp{beanCP.attribute2[0]}
</td>
<td class="heading">Complex Data type -
hashMapComplexcp{beanCP.attribute2[0]}{arrayCP[1]}.attribute2#</td>
</tr>
<tr class="row{position() mod 2 }">
<xsl:call-template name="showdata" />
</tr>
</table>
</body>
</html>
</xsl:template>
<xsl:template name="showdata">
<td><xsl:value-of select="$beanCP.attribute1"/></td>
<td><xsl:value-of select="$arrayCP__LB__0__RB__"/></td>
<td><xsl:value-of select="$beanCP__D__attribute2__LB__0__RB__"/></td>
<td><xsl:value-of select="$hashMapcp__LC__Q__key1__Q__RC__"/></td>
<td><xsl:value-of
select="$hashMapcp__LC__beanCP__D__attribute2__LB__0__RB__RC__"/></td>
<td><xsl:value-of
select="$hashMapComplexcp__LC__beanCP__D__attribute2__LB__0__RB__RC__LC
__arrayCP__LB__1__RB__RC__D__attribute2__H__"/></td>

```

```
</xsl:template>  
</xsl:stylesheet>  
<!-- Copyright 2007 Hewlett Packard Development Company, L.P. -->
```

## UpdateBean

`com.hp.ov.activator.mwfm.component.builtin.UpdateBean`

This node updates an inventory bean object in memory; i.e. the object is not stored in the inventory database after being updated.

**Table 4-129 UpdateBean Parameters**

Name	Required	Description	Default	Type
<i>bean_object</i>	Yes	Name of the variable holding the inventory bean object to be updated.	None	Object
<i>key_field0</i> , <i>key_field1</i> ... <i>key_fieldN</i>	Yes	Name of a key in the JavaBean that is updated. The parameter must be repeated for all attributes in the JavaBean being updated.  Since the object's primary key is implicitly passed to this node through the <i>bean_object</i> parameter there is no need for specifying the primary key	None	String
<i>key_value0</i> , <i>key_value1</i> , <i>key_valueN</i>	Yes	Used in conjunction with the <i>key_field</i> attributes to specify the values of the individual attributes in the JavaBean	None	Any
<i>bean_variable</i>	Yes	Name of the variable where the created JavaBean instance is returned.	None	Object

**Example 4-101 UpdateBean - use in the workflow**

This example updates in memory an inventory object representing a UNIX user.

```
<Process-Node>
  <Name>UpdateUnixUser</Name>
  <Description>Update a UNIX user</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.UpdateBean
    </Class-Name>
    <Param name="bean_object" value="user" />
    <Param name="key_field0" value="constant:home" />
    <Param name="key_value0" value="new_home_directory" />
  </Action>
</Process-Node>
```



## UpdateCustomAttributesNNMNode

`com.hp.ov.activator.mwfm.component.builtin.nnmrequest.UpdateCustomAttributesNNMNode`

The node supports the operation `updateCustomAttributes` (both in `NodeBeanService` and `InterfaceBeanService`). The `bean_type` parameter must be specified in order to determine which of the two available operations will be actually invoked..

**Table 4-130 UpdateCustomAttributesNNMNode Parameters**

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the NNMi module used to connect to a specific NNMi server	None	String
<i>bean_type</i>	Yes	The name of a bean type. The bean type can have one of the following values: NNM_NODE_BEAN NNM_INTERFACE_BEAN	None	String
<i>bean_id</i>	Yes	The the identifier of the bean whose custom attributes will be updated.	None	String
<i>action</i>	Yes	The parameter determines which operation is to be performed: The value can be either "ADD" or "REMOVE".	None	String
<i>custom_attribute_name0</i> , <i>custom_attribute_name1</i> , .... <i>custom_attribute_nameN</i>	No	Custom attribute name	None	String
<i>custom_attribute_value0</i> , <i>custom_attribute_value1</i> , .... <i>custom_attribute_valueN</i>	No	Custom attribute value	None	String
<i>custom_attribute_map</i>	Yes, If no <i>custom_attribute_name</i> is specified	The key-value pair set within the Map will be the name-value list of custom attributes to be updated	None	Object

## UpdateInProgress

`com.hp.ov.activator.mwfm.component.builtin.UpdateInProgress`

The node updates the value of the service-instance parameter called `IN_PROGRESS` for a given service identifier.

**Table 4-131** UpdateInProgress Parameters

Name	Required	Description	Default	Type
<i>service_id</i>	Yes	Unique identifier that the technical parameters are bound to.	None	String
<i>db</i>	No	Name of the database pluggable module to be used. Defaults to <code>db</code>	"db"	String
<i>status</i>	Yes	New status value to set the <code>IN_PROGRESS</code> service_instance_parameter to. This is specified as a constant string. You can use any set of strings for the status values that you want.	None	String

**Example 4-102** UpdateInProgress - use in the workflow

```
<Process-Node>
  <Name>Update IN_PROGRESS</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.UpdateInProgress
    </Class-Name>
    <Param name="service_id" value="customer_id"/>
    <Param name="status" value="available"/>
  </Action>
</Process-Node>
```

## UpdateInventory

`com.hp.ov.activator.mwfm.component.builtin.UpdateInventory`

The node creates or updates instances in the inventory. It sets `RET_VALUE` to 0 if successful, and to 1 if create or update fails. The supplied value of the primary key determines whether the node creates or updates instances in the inventory. If the key already exists, the specified attributes are modified otherwise a new instance is created in the inventory.

Values can be passed to an inventory object either by specifying a list of `key_field/key_value` pairs or by passing an object containing the inventory bean.

**Table 4-132 UpdateInventory Parameters**

Name	Required	Description	Default	Type
<i>db</i>	No	Name of the database module to be used.	“db”	String
<i>bean</i>	Yes	Name of the JavaBean class that is used for storing the data.	None	String
<i>bean_object</i>	No	The name of the variable containing the inventory bean object to be stored in the inventory.	None	Object
<i>key_field0</i> , <i>key_field1</i> ... <i>key_fieldN</i>	No	Name of a key in the JavaBean that is updated or created. The parameter must be repeated for all attributes in the JavaBean being updated or initially assigned. Note that when a JavaBean is updated the primary key must always be present in the list of keys, even if it is not updated. Note that the <i>key_fields</i> may be case packet variables.	None	String
<i>key_value0</i> , <i>key_value1</i> , <i>key_valueN</i>	No	Used in conjunction with the <i>key_field</i> attributes to specify the new value of the individual attributes in the JavaBean	None	Depends on the bean
<i>bean_variable</i>	No	Name of the variable where the created/updated JavaBean instance is returned.	None	Object
<i>strict_create</i>	No	When set to “true” the node will run in “strict create” mode which means that the node will fail if a bean with the specified key does already exist.  Can not be used together with the <i>strict_update</i> parameter.	false	Boolean

**Table 4-132**      **UpdateInventory Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>strict_update</i>	No	When set to “true” the node will run in “strict update” mode which means that the node will fail if a bean with the specified key does not exist.  Can not be used together with the <i>strict_create</i> parameter.	false	Boolean
<i>store_audit</i>	No	If audit is enabled in the Workflow Manager’s configuration file as well as in the Inventory Bean’s XML resource definition file an audit record will be written each time this node is executed.  To disable audit for the node set this parameter to “false”.	true	Boolean

**Example 4-103 UpdateInventory - use in the workflow**

This example uses the `UpdateInventory` node to modify the state of a port. The primary key is held in the variable `port_id` and the new state variable is denoted by the text *Exclusive*.

```
<Process-Node>
  <Name>PortUsage=Exclusive</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.UpdateInventory
    </Class-Name>
    <Param name="bean" value="com.hp.ov.activator.example.Port"/>
    <Param name="key_field0" value="ElementComponentId"/>
    <Param name="key_value0" value="port_id"/>
    <Param name="key_field1" value="UsageState"/>
    <Param name="key_value1" value="Exclusive"/>
  </Action>
</Process-Node>

<Case-Packet>
  <Variable name="port_id" type="String"/>
</Case-Packet>
```

The following example shows how to create a new instance in the inventory of the JavaBean `L2VPN`, which is a service instance for Layer 2 VPN. Note that the `comments` attribute is specified as `constant:comments` because it has the same name as the case-packet variable.

```
<Process-Node>
  <Name>Create L2 VPN</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.UpdateInventory
    </Class-Name>
    <Param name="bean" value="com.hp.ov.activator.example.L2VPN"/>
    <Param name="bean_variable" value="vpn_obj"/>
    <Param name="key_value0" value="service_id"/>
    <Param name="key_field0" value="ServiceId"/>
    <Param name="key_field1" value="CustomerId"/>
    <Param name="key_value1" value="customer_id"/>
    <Param name="key_field2" value="constant:comments"/>
    <Param name="key_value2" value="comments"/>
    <Param name="key_field3" value="Name"/>
    <Param name="key_value3" value="vpn_name"/>
    <Param name="key_field4" value="ActivationDate"/>
    <Param name="key_value4" value="date"/>
  </Action>
</Process-Node>

<Case-Packet>
  <Variable name="service_id" type="String"/>
  <Variable name="customer_id" type="String"/>
  <Variable name="comments" type="String"/>
  <Variable name="vpn_name" type="String"/>
  <Variable name="date" type="String"/>
</Case-Packet>
```

## UpdateServiceInstance

`com.hp.ov.activator.mwfm.component.builtin.UpdateServiceInstance`

The node updates the service-instance repository to set new values for the desired technical parameters tied to a given unique service identifier.

**Table 4-133 UpdateServiceInstance Parameters**

Name	Required	Description	Default	Type
<i>service_id</i>	Yes	Unique identifier to which the technical parameters are bound.	None	String
<i>db</i>	No	Name of the database module to use.	"db"	String
<i>variable0</i> , <i>variable1</i> , ...	Yes	Names of the different technical parameters to update. You must specify at least <i>variable0</i> .	None	String

**Example 4-104 UpdateServiceInstance - use in the workflow**

This example updates several technical parameters that are tied to a customer identifier.

```
<Process-Node>
  <Name>Update technical inventory</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.UpdateServiceInstance
    </Class-Name>
    <Param name="service_id" value="customer_id"/>
    <Param name="db" value="db"/>
    <Param name="variable0" value="web_domain"/>
    <Param name="variable1" value="group"/>
    <Param name="variable2" value="homedir"/>
    <Param name="variable3" value="ipaddress"/>
    <Param name="variable4" value="logdir"/>
    <Param name="variable5" value="login"/>
    <Param name="variable6" value="machine"/>
    <Param name="variable7" value="password"/>
    <Param name="variable8" value="port"/>
    <Param name="variable9" value="pre_domain"/>
    <Param name="variable10" value="rootdir"/>
    <Param name="variable11" value="uid"/>
  </Action>
</Process-Node>
```

## UpdateUCMDBCIsAndRelations

`com.hp.ov.activator.mwfm.component.builtin.UpdateUCMDBCIsAndRelations`

The updateUCMDBCIsAndRelations node will update the specified CIs and Relations in the uCMDB.

This node can update multiple CIs and Relations in a single request. The node throws a UCMDBException in case there is an error while processing the request.

**Table 4-134**

### UpdateUCMDBCIsAndRelations Parameters

Name	Required	Description	Default	Type
<i>module_name</i>	Yes	The name of the UCMDBRequestModule to be used	None	String
ci_id0 ci_id1... ci_idN	Yes (At least one is mandatory if no relations are specified. Not mandatory if relations are specified)	UCMDB Id of the CI which needs to be updated. A single CI can have multiple properties. This can be specified by giving the same CI Id again.	None	String
ci_type0 ci_type1... ci_typeN	Yes (If ci_id has been specified)	Type of the CI. It can be any type defined in uCMDB	None	String
ci_prop_name0 ci_prop_name1 ... ci_prop_name N	No	Name of the property to be associated with the CI	None	String

**Table 4-134 UpdateUCMDBCIAndRelations Parameters**

Name	Required	Description	Default	Type
ci_prop_value0 ci_prop_value1 ... ci_prop_value N	No	Value of the property name specified earlier. In case the property type is StringList or IntList then the property values can be a list of values. This can be specified by separating the values with the # character. The ci_prop_value can also be specified as a case-packet variable. In case the property type is a StringList or an IntList then the case-packet variable has to be of type Object, Internally it can contain either a String[] or a List	None	String
ci_prop_type0	No	The type of the property. This can take the following values:  String Byte Integer Long Float Double Boolean Date XML StringList IntList	None	String
rel_id0 rel_id1... rel_idN	Yes (Atleast one is mandatory if no CIs are specified. Not mandatory if CIs are specified)	UCMDB Id of the Relation which needs to be updated.	None	String
rel_type0 rel_type1... rel_typeN	Yes (If relation id has been specified)	Type of the Releation	None	String



**Table 4-134 UpdateUCMDBCIAndRelations Parameters**

Name	Required	Description	Default	Type
rel_end1_id1 rel_end1_id2... rel_end1_idN	Yes (If relation id has been specified)	End 1id of the relation. The ID of the CI at end 1 of the relation.	None	String
rel_end2_id1 rel_end2_id2... rel_end2_idN	Yes (If relation id has been specified)	End 2 id of the relation. The ID of the CI at end 2 of the relation.	None	String
rel_prop_name 0 rel_prop_name 1 .... rel_prop_name N	No	Name of the property to be associated with the Relation	None	String
rel_prop_value 0 rel_prop_value 1 ... rel_prop_value N	No	Value of the property name specified earlier In case the property type is StringList or IntList then the property values can be a list of values. This can be specified by separating the values with the # character The rel_prop_value can also be specified as a case-packet variable. In case the property type is a StringList or an IntList then the case-packet variable has to be of type Object, Internally it can contain either a String[] or a List	None	String
rel_prop_type0 rel_prop_type1 ... rel_prop_type N	No	The type of the property. This can take the following values:  String Byte Integer Long Float Double Boolean Date XML StringList IntList	None	String

**Table 4-134**      **UpdateUCMDBCIAndRelations Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
date_format	No	Specifies the format in which the ci_property_value and rel_prop_values have been defined in case the property type is Date. The date format can be specified using standard java conventions used while defining a date format (as in the SimpleDateFormat class). In case this parameter is not specified then the date format is taken as the default one for the current locale in which HPSA has been deployed.	System's Locale's date format is taken	String

## VariableMapper

`com.hp.ov.activator.mwfm.component.builtin.VariableMapper`

The node sets the value of case-packet variables based on templates. A template for variable mapping is a string that can have embedded references to other case-packet variables. For example, to construct the name of a home directory, you might want to append the user name to a fixed root path. The template might be `/home/%username%`. The `%varname%` syntax indicates the portions of the template that should be replaced.

The mappings can be specified in the workflow node parameters or can be placed in a template file. The template file can then be referenced from multiple workflows.

It is valid to specify both a `template_file` and individual variables to be mapped.

The node always maps a string value to the case-packet variable specified. If you use the node to copy the value of a field in a bean when that value is null, then the resulting string will have the value "null". If you need to copy an object, then use `Assign` node.

### See Also

- "Assign" on page 109

**Table 4-135** VariableMapper Parameters

Name	Required	Description	Default	Type
<i>template_file</i>	No	Name of a file that holds a list of mappings. The default path to find files is <code>\$ACTIVATOR_ETC/template_files</code> . You can specify an absolute path name.  Mappings in the file are each on a separate line and have the following syntax <code>var=template</code>	None	String
<i>name of a case-packet variable</i>	No	Value of the parameter is a template string for setting the new value of the indicated case-packet variable.	None	String
<i>use_solution_dir</i>	No	When set to "true", the nodes will read from <code>\$SOLUTION_ETC/template_files</code> instead of <code>\$ACTIVATOR_ETC/template_files</code> .	false	boolean

**Example 4-105** VariableMapper - use in the workflow

This example sets the value of the `homedir` variable to `/home/ravi` and the `password` variable to `raviPW` and assumes that the variable `login` has the value `ravi`.

```
<Process-Node disablePersistence="true">
  <Name>Map Values</Name>
  <Action>
    <Class-Name>
```

```
        com.hp.ov.activator.mwfm.component.builtin.VariableMapper
    </Class-Name>
    <Param name="homedir" value="/home/%login%" />
    <Param name="password" value"%login%PW" />
  </Action>
</Process-Node>
```

## WasPreviousNodeOK

`com.hp.ov.activator.mwfm.component.builtin.WasPreviousNodeOK`

The node tests whether the previous node was executed normally. If the previous node was processed correctly, the workflow follows the true branch otherwise it follows the false branch. The previous node was executed normally if the `RET_VALUE` has the value 0.

**Table 4-136** WasPreviousNodeOK Parameters

Name	Required	Description	Default	Type
<code>priv_ret_value</code>	No	This argument makes it possible to save the value of the system case-packet variable <code>RET_VALUE</code> for the previous node. This value may be necessary to handle errors later on.	None	Integer
<code>priv_ret_text</code>	No	This argument has the same functionality as the above apart from saving the <code>RET_TEXT</code> variable.	None	String

**Example 4-106** WasPreviousNodeOK - use in the workflow

This example checks whether the previous node in the workflow was processed correctly. If the node was processed correctly, the next node is `PrintOkMessage` otherwise the `PrintFailedMessage` node is next. The `RET_VALUE` and `RET_TEXT` variables from the previous node are saved in the case-packet variables `last_ret_value` and `last_ret_text`

```
<Rule-Node disablePersistence="true">
  <Name>WasPreviousNodeOK</Name>
  <Description>Check the previous node</Description>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.WasPreviousNodeOK
    </Class-Name>
    <Param name="priv_ret_text" value="last_ret_text" />
    <Param name="priv_ret_value" value="last_ret_value" />
  </Action>
  <True-Next-Node>PrintOkMessage</True-Next-Node>
  <False-Next-Node>PrintFailedMessage</False-Next-Node>
</Rule-Node>
```

## WriteCasePacket

`com.hp.ov.activator.mwfm.component.builtin.WriteCasePacket`

The node provides a way to write the contents of a case-packet to a file or to a sender module. This is typically used for testing.

**Table 4-137** WriteCasePacket Parameters

Name	Required	Description	Default	Type
<i>file</i>	Yes, if sender is not specified	Name of the file to which the case-packet is written. The value of this parameter can be a case-packet variable that contains the name of the file, or can be a constant (specified as <code>constant:X</code> where <i>X</i> is the name of the file).  If the path name to the file is not an absolute path, the file is created relative to <code>\$(ACTIVATOR_VAR)/tmp</code>	None	String
<i>sender</i>	Yes, if <i>file</i> is not specified	Module name of a sender module.	None	String

## WriteDataToDatabase

`com.hp.ov.activator.mwfm.component.builtin.WriteDataToDatabase`

The node writes or updates data in the DATABASE\_MESSAGE table in the database.

The first time data is written the message\_url is not required. A message id will be returned as an identifier in the output parameter output\_value.

When updating or appending data the message\_url must contain the message id.

The data to be written must be given in the parameter message\_data.

If an identifier must be written to the identifier column this information can be provided in the parameter identifier.

If the node is used to update existing data an optional parameter "data\_position" is provided.

**Table 4-138 WriteDataToDatabase Parameters**

Name	Required	Description	Default	Type
<i>message_url</i>	No	Name of the case packet variable holding the message id. The syntax is db:<message_id>	None	String
<i>identifier</i>	No	Name of the case packet variable holding the identifier. The information will be written in the identifier column	None	String
<i>output_value</i>	Yes	Case packet variable holding the returned message id. The syntax is db:<message_id>.	None	String
<i>message_data</i>	Yes	Name of the case packet variable containing the message to be written.	None	Object or String
<i>data_position</i>	No	The value of this parameter is the ordinal position from where the data is written.	None	String

**Example 4-107 Write Data**

```
<Process-Node>
  <Name>WriteDataToDatabase</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.WriteDataToDatabase
    </Class-Name>
    <Param name="message_data" value="saveData" />
    <Param name="output_value" value="newMessageId" />
  </Action>
</Process-Node>
```

**Example 4-108 Update Data**

```
<Process-Node>
  <Name>WriteDataToDatabase2</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.WriteDataToDatabase
    </Class-Name>
    <Param name="data_position" value="31" />
    <Param name="message_data" value="saveData" />
    <Param name="message_id" value="newMessageId" />
    <Param name="output_value" value="newMessageId" />
  </Action>
</Process-Node>
```



## XMLMapper

`com.hp.ov.activator.mwfm.component.builtin.XMLMapper`

The node maps fields from an XML message to case-packet variables.

To set each case-packet variable, there is a specification for finding the proper element of the XML message. The specification is similar to a directory path. Each element in the tree is separated with the slash character.

If you want to test an attribute of an element, put a pipe (|) character at the end, and append the name of the attribute to retrieve (Msg/Body|ID). It is valid to specify both a `template_file` and individual variables to be mapped.

The XMLMapper node supports the use of the hash character (#) for inserting lists. When # is used together with a numeric value (#<number>), reference is made to a specific entry in a list. For instance, (#5) fetches the 5th entry from a list.

By default, the XMLMapper node raises an exception if it cannot find specified tags for a mapping. You can indicate to the node whether it should ignore such cases or not. If the parameter `ignore_missing_tags` has a value of “true” and the tags for one of the specified mappings cannot be found in the XML file, an exception will not be raised, but the variable intended to receive the value will be set to a default value (“ ” for Strings, 0 for Integer and Float, “false” for Boolean). If this is the case, the XMLMapper node sets `RET_VALUE` to 1 to indicate that some values were not mapped.

The `RET_VALUE` variable is updated with a value of 0 if reading and parsing the XML file are successful or a value of 1 in case of an error.

The XMLMapper node uses a special syntax to access multiple tags of the same name. For example, if the following statement appears in the XML file:

```
<Parameter name="option_type" value="/msg/body/option#/type"/>
```

The following code would be expected:

```
<msg>
  <body>
    <option>
      <type>A</type>
    </option>
    <option>
      <type>B</type>
    </option>
    <option>
      <type>C</type>
    </option>
  </body>
</msg>
```

This would cause the `option_type` variable (a variable of type `Object` that was previously created) to contain an array of three values { "A", "B", "C" }.

It is also possible to extract information from an XML message which does not have a root element if the validation parameter is set to “false”. For example, for the following XML message:

```
<A><B>b1</B><C>c1</C></A>
<A><B>b2</B></A>
<A><C>c3</C></A>
<A><B>b4</B></A>
```

With the mapping:

```
<Parameter name="varB" value="A/B"/>
<Parameter name="varC" value="A/C"/>
```

Gives the following two objects:

```
varB=Object[]{"b1","b2","b4"}
varC=Object[]{"c1","c3"}
```

The XMLMapper node supports the functionality to extract XML parts of the XML message. Using the above example and changing the mapping to the following:

```
<Parameter name="varB" value="A"/>
```

Gives the following object:

```
varA= Object[]{"<B>b1</B><C>c1</C>","<B>b2</B>","<C>c3</C>","<B>b4</B>"}
```

The XML to be mapped is specified using the action parameter *xml\_url*. This can be a file (an absolute path or a filename relative to \$ACTIVATOR\_VAR), or message id that refers to message stored in the database. Alternatively, the XML can also be specified using the action parameter *xml\_var*.

**Table 4-139 XMLMapper Parameters**

Name	Required	Description	Default	Type
<i>xml_url</i>	Yes (if not <i>xml_var</i> is specified)	The file to be mapped (an absolute path, or a filename relative to \$ACTIVATOR_VAR), or the message id that refers to a message stored in the database, or a data string. The syntax is file:<file path>, data:<string>, or db:<message_id>.	None	String
<i>template_file</i>	No	Name of a file that holds a list of mappings. The default path to find files is: \$ACTIVATOR_ETC/template_files. You can specify an absolute path name. The mappings in the file are each on a separate line and have the following syntax: var = template	None	String
<name of a case-packet variable>,...	No	Value of the parameter is a template string for setting the new value of the indicated case-packet variable.	None	String
<i>validate</i>	No	By default, the XML file is validated against its declared DTD. Set this parameter value=false if validation is not to be carried out.	None	Boolean

**Table 4-139 XMLMapper Parameters (Continued)**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>ignore_missing_tags</i>	No	Indicates whether XML tags that are not found will cause an exception to be raised or not. Note that the parameter is ignored when inserting a list. When used together, the operators # and   must be grouped so that # comes last. For example, an array with values 1 and 2 can be fetched from <pre>&lt;A&gt;&lt;B b="1" /&gt;&lt;B b="2" /&gt;&lt;/A&gt;</pre> by typing A/B b# (not A/B# b)	"false"	Boolean
<i>xml_var</i>	Yes (if not xml_url is specified)	The name of the case-packet variable containing the XML message.	None	String
<i>preserve_variable_index</i>	No	Indicating if wheater an optional element having been missed out should be represented by a null value	False	Boolean
<i>use_solution_dir</i>	No	When set to "true", the nodes will read from \$SOLUTION_ETC/template_files instead of \$ACTIVATOR_ETC/template_files.	false	Boolean

### Example 4-109 XMLMapper - use in the workflow

This example parses an incoming XML file to set the value of the variables `custid` and `uid`. The case-packet message\_url contains the file path or message id.

```
<Process-Node>
  <Name>XML mapper</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.XMLMapper
    </Class-Name>
    <Param name="xml_url" value="message_url" />
    <Param name="custid" value="msg/header/customer_id" />
    <Param name="uid" value=msg/body/login|uid" />
  </Action>
</Process-Node>
```

### Example 4-110 XMLMapper - incoming message to be parsed

This example of an XML message will set the `custid` to 12345 and the `uid` to 522.

```
<msg>
  <header>
    <customer_id>12345</customer_id>
    <message_id>93456</message_id>
    <service_name>OVACT_ActivateWeb</service_name>
  </header>
  <body>
    <domain>storactive.cnd.hp.com</domain>
    <login uid="522" group="users">mylogin0</login>
    <password>mypass01</password>
  </body>
</msg>
```

### XML Namespaces

The XMLMapper node has a basic understanding of XML namespaces. To specify a namespace, separate the namespace from the element with a colon (:):

```
<soap:Envelope>
```

To select the above element, use:

```
/soap:Envelope/...
```

### Example 4-111 XMLNamespaces

```
<?xml version="1.0" encoding="UTF-8"?>
  <Document
    xmlns:ns1="http://www.hp.com/ns1"
    xmlns:ns2="http://www.hp.com/ns2">
    <ns1:Test>
      <ns1:value>test</ns1:value>
    </ns1:Test>
    <Underscore>_</Underscore>
    <ns2:Test>
      <ns2:value>succeeded</ns2:value>
    </ns2:Test>
  </Document>
```

In the above example 'test' value can be extracted by the following path:

```
/Document/ns1:Test/ns1:value
```

The 'succeeded' value can be extracted by the following path:

```
/Document/ns2:Test/ns2:value
```

The 'underscore' value can be extracted by the following path:

```
/Document/Underscore
```

## XMLParser

`com.hp.ov.activator.mwfm.component.builtin.XMLParser`

The node allows validating an XML against a DTD or W3C Schema and also allows retrieval of data from the xml using the W3C XPath notation. The node will map values corresponding to the XPaths to the specified case-packet variables.

The XMLParser node will only support XPaths with absolute paths conforming to XPath 1.0 specification. The XMLParser does not support specifying of paths in relative format nor does the node support the XPath AND operator(|).

This node can read an xml from a case-packet variable, or a file or from a database message. In case the xml is read from a case-packet variable then the `xml_var` parameter needs to be specified and in case the xml is read from a file or a db message id then the `message_url` parameter has to be specified. Specifying either one of these parameters is mandatory. However both these parameters cannot be specified together.

The XMLParser node also supports parsing an xml which does not have root nodes. However to enable this, the "validate" parameter must be set to false. XMLParser node also supports an XML with namespaces. To parse an xml with namespaces the namespace prefixes and the namespace urls used in the XML needs to be specified using the `namespace_prefix` and the `namespace_url` parameters. Multiple prefixes and namespaces can be specified.

The input xml can either be validated against a DTD or a schema. The parameter "validate" determines if the input xml needs to be validated or not. The value of this parameter is set to "true" by default. The value of the parameter `definition_language` determines whether the validation has to be against a DTD or a W3C Schema. The default value of this parameter is "W3CSchema". The schema or the DTD file can either be embedded as part of the input XML or specified separately using the `definition_url` parameter. In case schema/dtd has been specified in the `definition_url` parameter and also embedded in the document, then the xml is validated against the file specified by the `definition_url` parameter.

The XMLParser allows retrieval of data from the XML using the XPath notation. The result retrieved corresponds to the type of case-packet variable defined. In case the case-packet variable is of type Object, then the result retrieved will be a String array of values of all the nodes matching the XPath. In case the case-packet variable is of any other type then the value of the first node matching the XPath will be returned.

To set each case-packet variable, an XPath corresponding to the element or attribute whose value needs to be fetched has to be given.

Consider the XML:

```
<msg>
  <body name="item1">
    <option name="1">
      <type>A</type>
      <colour>Red</colour>
    </option>
    <option>
      <type>B</type>
    </option>
    <option name="2">
      <type>C</type>
      <colour>Black</colour>
    </option>
  </body>
```

```

<body name="item2">
  <option name="3">
    <type>D</type>
    <colour>Blue</colour>
  </option>
  <option name="2">
    <type>F</type>
    <colour>Yellow</colour>
  </option>
</body>
</msg>

```

In case the value of the first type node needs to be fetched then the XPath can either be:

```
/msg/body/option/type
```

Or

```
/msg/body/option[1]/type
```

It is to be noted that in case of XPath notations the index starts from 1 and not 0. In both cases the case-packet variable needs to be of type String. The value returned will be string "A".

In case the XMLParser cannot find specified tags for an XPath, then by default, the node returns default values, the variable intended to receive the value will be set to a default value ( " " for Strings, 0 for Integer and Float, "false" for Boolean). In case an exception needs to be raised when a tag is not found then you can set the ignore\_missing\_tags parameter to false. The default value of this parameter is true. However the ignore\_missing\_tags parameter is not considered if the case-packet variable is of type Object.

In case multiple values need to be fetched, then the XPath /msg/body/option/type needs to be given with the case-packet variable set as Object. The value returned will be the string array {"A", "B", "C", "D", "E", "F"}.

In case the type of the case-packet variable is Object and the xml does not have the element corresponding to the XPath then an "" is placed in the string array as a placeholder for the element. For example, the element colour is not present in the second option node of the example. When the XPath has been specified as /msg/body/option/colour then the value returned will be a string array with the values {"Red", "", "Black", "Blue", "", "Yellow"}.

Attributes can be fetched by prefixing them with the @ symbol. The value of an attribute can be fetched by using the XPath /msg/body/option/@name.

XPaths can also be specified in a template file. The way to specify a variable in a template file is as follows <variablename>=<xpath>. For example if the case-packet variable name is option\_type and the xpath that need to be defined is /msg/body/option/type then the same can be defined as option\_type=/msg/body/option/type. It is to be noted that the case-packet variable name "option\_type" needs to be defined in the workflow.

It is valid to specify both a template\_file and individual variables to be mapped. In case the same variable name has been specified both in the template-file and the individual variable, then the XPath defined against the individual variable takes precedence.

**Table 4-140 XMLParser Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>	<b>Type</b>
<i>xml_var</i>	Yes (if message_url is not given)	The case packet variable which contains the xml	None	String
<i>message_url</i>	Yes (yes if xml_url is not given)	The file to be mapped (an absolute path, or a filename relative to \$ACTIVATOR_VAR), or the message id that refers to a message stored in the database, or a data string. The syntax is file:<file path>, data:<string>, or db:<message_id>.	None	String
<i>validation</i>	No	Indicates whether the xml needs to be validated against a DTD/Schema	True	Boolean
<i>definition_language</i>	No	Indicates the type of language used to validate the xml. Possible values are DTD and W3CSchema	W3CSchema	String
<i>definition_url</i>	No	The url to locate the schema or the DTD(an absolute path, or a filename relative to \$ACTIVATOR_ETC/config), The syntax is file:<file path>.	None	String
<i>template_file</i>	No	Name of a file that holds a list of mappings. The default path to find files is: \$ACTIVATOR_ETC/template_files. You can specify an absolute path name.  The mappings in the file are each on a separate line and have the following syntax: var = template	None	String
<i>use_solution_dir</i>	No	When set to "true", the nodes will read from \$SOLUTION_ETC/template_files instead of \$ACTIVATOR_ETC/template_files.	false	Boolean



**Table 4-140 XMLParser Parameters (Continued)**

Name	Required	Description	Default	Type
variable0 variable1 ... variableN	No	One or more case-packet variables whose values are being requested. In case any variable name matches with the one specified in a template file as well, then the XPath specified as variable in the node takes precedence over the one defined in the template-file.	None	String/object
xpath0 xpath1 ... xpathN	No	The XPath that should be fetched against each corresponding variable. In case namespace_prefix values have been specified, then the xpaths should also contain the namespaces prefixed according to xpath conventions.	None	String
namespace_prefix0 namespace_prefix1 ... namespace_prefixN	No	The namespace prefixes that are used in the input XML	None	String
namespace_url0 namespace_url1 ... namespace_urlN	Yes (if namespace_prefix has been specified.)	In case the XML contains namespaces, the namespace urls that map to the specified prefixes should be specified in this parameter.		

**Table 4-140 XMLParser Parameters (Continued)**

Name	Required	Description	Default	Type
<i>ignore_missing_tags</i>	No	Indicates whether XML tags that are not found will cause an exception to be raised or not. Note that the parameter is ignored when inserting a list. When used together, the operators # and   must be grouped so that # comes last. For example, an array with values 1 and 2 can be fetched from <A><B b="1" /><B b="2"></A> by typing A/B b# (not A/B# b)	"false"	Boolean

---

## Handlers

This section describes the handlers supplied with Service Activator. Each of these handlers is suitable as an error handler or an end handler. Each handler is implemented by a Java class. The name of the handler is the name of the class that implements it. Note, however, that it is the full name (including the package name) that uniquely identifies the handler. All of the built-in handlers shipped with Service Activator are from the same package (`com.hp.ov.activator.mwfm.component.builtin`).

## ComposeMessageHandler

`com.hp.ov.activator.mwfm.component.builtin.ComposeMessageHandler`

This handler behaves identically to the ComposeMessage node; i.e. the handler can compose a message string based on a template and a number of case-packet variables..

### See also

- “ComposeMessage” on page 114

**Table 4-141 ComposeMessageHandler Parameters**

Name	Required	Description	Default
<i>template_file</i>	Yes if <i>template_var</i> is not used	Name of the file in which the template is to be found. The value of this parameter can be a case-packet variable that contains the name of the file, or can be a constant (specified as constant:X where X is the name of the file).  The file is expected to exist in one of the directories \$SOLUTION_ETC/template_files or \$ACTIVATOR_ETC/template_files, depending on the value of the <i>use_solution_dir</i> parameter.	None
<i>template_var0</i> , <i>template_var1</i> , ... <i>template_varN</i>	Yes if <i>template_file</i> is not used	Name of a case-packets available that contains the template strings.	None
<i>user_solution_dir</i>	No	When set to "true", the handler will read from \$SOLUTION_ETC/template_files instead of \$ACTIVATOR_ETC/template_file	false
<i>output_file</i>	Yes if <i>output_var</i> is not used	Name of the file to which the composed message is to be written. The value of this parameter can be a case-packet variable that contains the name of the file, or it can be a constant (specified as constant:X where X is the name of the file).  If the path name to the file is not an absolute path, the file is created relative to \$ACTIVATOR_VAR/tmp	None

**Table 4-141**      **ComposeMessageHandler Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>
<i>output_var</i>	Yes if <i>output_file</i> is not used	Name of a case-packet variable in which the composed message is placed.	None

## MultiAssignHandler

`com.hp.ov.activator.mwfm.component.builtin.MultiAssignHandler`

This handler behaves identically to the MultiAssign node; i.e. the handler can assign values to multiple case-packet variables..

### See also

- “MultiAssign” on page 216

**Table 4-142 MultiAssignHandler Parameters**

Name	Required	Description	Default
<i>Variable0,</i> <i>Variable1,</i> ... <i>VariableN</i>	Yes	Case-packet variables to be set.	None
<i>Value0,</i> <i>Value1,</i> ... <i>ValueN</i>	Yes	New value to set for the variable. It can be a case-packet variable or a constant (specified as <code>constant:X</code> where X is the constant).	None

## DoNothingHandler

`com.hp.ov.activator.mwfm.component.builtin.DoNothingHandler`

The handler does nothing except for logging a message. The parameters of this handler are the same as those of the `DoNothing` node.

### See also

- “DoNothing” on page 140

**Table 4-143**

### DoNothingHandler Parameters

Name	Required	Description	Default
<i>message</i>	No	The message to be logged.	None

## PutMessageHandler

`com.hp.ov.activator.mwfm.component.builtin.PutMessageHandler`

The handler puts a message on a message queue. The messages will be persisted in the database. Optionally, the messages can also be associated with a solution. Since roles cannot be associated with handlers the workflow's default role will be used for posting the message.

---

**NOTE** If the message is more than 4000 bytes the message will be truncated to 4000 bytes.

---

**Table 4-144 PutMessageHandler Parameters**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>
<i>queue</i>	Yes	Queue where the message is left. This parameter can either be a constant or a case-packet variable. Spaces are not allowed.	None
<i>message</i>	Yes	Message to be printed. Any %s symbols appearing in the string are replaced by consecutive paramN parameters. Functions similar to printf in the C programming language.	None
<i>param0</i> , <i>param1</i> , ... <i>paramN</i>	No	If the message contains any %s symbols, the first one is replaced by the value of the variable indicated by param0, the next by param1, and so on. The variables can be of any type. However, their values are converted to strings.	None
<i>service_id</i>	No	The Service Identifier value used for associating the message with a service or a solution.	None
<i>order_id</i>	No	The order Identifier value used for associating the message with a service or a solution.	None
<i>type</i>	No	The type value of the workflow	None
<i>state</i>	No	The state value of the workflow	None



## ReleaseResourceHandler

`com.hp.ov.activator.mwfm.component.builtin.ReleaseResourceHandler`

The handler releases resources that have been reserved within a workflow. It would typically be used as an error handler. If the workflow reserves a resource but terminates abnormally before the resource is actually put into use, it might be appropriate to release the resource before the workflow completes. This handler can be used to release resources contained in specifically listed variables (use the parameters `variable0...variableN`) or to release all of the resources currently held in the `RESERVATIONS` variable. The parameters of this handler are the same as those of the `ReleaseResource` node.

### See Also

- “ConfirmResourceReservation” on page 118
- “ReleaseResource” on page 276
- “ReserveResource” on page 281

**Table 4-145 ReleaseResourceHandler Parameters**

Name	Required	Description	Default
<i>db</i>	No	Database module to use in order to perform the update.	“db”
<i>variable0, variable1... variableN</i>	No	Name of a case-packet variable that holds the resource to be released. If no variables are specified then all reserved resources in the <code>RESERVATIONS</code> variable are released.	None

## SendMessageHandler

`com.hp.ov.activator.mwfm.component.builtin.SendMessageHandler`

This handler implement the same functionality as the SendMessage node; i.e. it sends messages using a SenderModule. It can use any module that implements the SenderModule interface (for instance, the SocketSenderModule or the JMSSenderModule).

The message to be sent can come from a case-packet variable, or from a file, or from the database.

When the handler completes, the value of the built-in case-packet variable RET\_VALUE is set to 0 if the message was properly enqueued and to 1 if not.

### See also

- “SendMessage” on page 291

**Table 4-146** SendMessageHandler Parameters

Name	Required	Description	Default
<i>sender</i>	Yes	Name of the Workflow Manager module that will send this message.	None
<i>message_var</i>	Yes if <i>message_url</i> is not used	Name of a case-packet variable that contains the message to be sent.	None
<i>message_url</i>	Yes if <i>message_var</i> is not used	Name of a case-packet variable that contains the name of the file or a message id representing a row in DATABASE_MESSAGE containing the message. The syntax is <code>db:message_id</code> or <code>file:filepath</code> . A constant file name or message id can also be specified.	None

## SyncHandler

`com.hp.ov.activator.mwfm.component.builtin.SyncHandler`

The handler ensures that a child workflow synchronizes with its controller workflow before the workflow completes. Rather than placing a `Sync` node explicitly in every path of your workflow, you can use the `SyncHandler` to ensure that irrespective of the path the workflow follows, or even if the workflow terminates abnormally, the child workflow synchronizes with its parent workflow.

If you use the `SyncHandler` as an error handler, you can also specify a parameter to indicate what to do with the exception message. The value of the `exception_destination` parameter indicates the name of the case-packet variable in the target workflow that should receive the exception message.

The handler will ensure that the synchronization is done even in case where the child workflow tries to synchronize before the parent workflow enters the `AskFor` node. And irrespective of the parent waiting condition, the children workflows will not be parked in any queue. The sync module will record the response with the parent and the children will go on to completion.

### See Also

- “AskFor” on page 104

**Table 4-147** **SyncHandler Parameters**

Name	Required	Description	Default
<i>job_id</i>	Yes	Name of a case-packet variable that contains the <code>job_id</code> of the workflow waiting to synchronize.	None
<i>queue</i>	Yes	Name of the queue on which the workflow is waiting, specified as constant or a case packet variable that contains the queue name.	None
<i>variable0,</i> <i>variable1...</i> <i>variableN</i>	Yes	Names of the case-packet variables to be passed to the waiting workflow.	None
<i>destination0,</i> <i>destination1...</i> <i>destinationN</i>	Nop	Names of the case-packet variables in the parent workflow that are waiting to receive the matching variable from this workflow. By default, the variables are passed to variables of the same name in the waiting workflow. Destination parameters can be specified selectively for some or all of the indicated variables.	None

**Table 4-147**      **SyncHandler Parameters (Continued)**

<b>Name</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>
<i>exception_destination</i>	No	The value of the <i>exception_destination</i> parameter indicates the name of a case-packet variable in the target workflow that should receive the exception message.	None
<i>sync</i>	No	Boolean value to indicate if a synchronization should be performed or not. Default value is true (synchronization will be performed).	true

## VariableMapperHandler

`com.hp.ov.activator.mwfm.component.builtin.VariableMapperHandler`

This handler implement the same functionality as the VariableMapper node; i.e. it set the values of one or more case-packet variables based on templates and other case-packet variable values.

### See also

- “VariableMapper” on page 323

**Table 4-148** VariableMapperHandler Parameters

Name	Required	Description	Default
<i>template_file</i>	No	Name of a file that holds a list of mappings. The directory to find files is <code>\$SOLUTION_ETC/template_files</code> or <code>\$ACTIVATOR_ETC/template_files</code> , depending on the value of the <code>use_solution_dir</code> parameter. You can specify an absolute path name. Mappings in the file are each on a separate line and have the following syntax <code>var=template</code>	None
<i>name of case-packet variable</i>	No	Name of a case-packets available that contains the template strings.	None
<i>user_solution_dir</i>	No	When set to “true”, the handler will read from <code>\$SOLUTION_ETC/template_files</code> instead of <code>\$ACTIVATOR_ETC/template_file</code>	false



---

## 5 **Configuring the Workflow Manager**

The Workflow Manager has many parameters that can be used to alter its behavior and tune its performance. Additionally, there are various Workflow Manager modules that can be configured to extend the capabilities of the Workflow Manager. These are all configured in the `$ACTIVATOR_ETC/config/mwfm.xml` file.

## Setting the Workflow Manager Parameters

During installation, all of the Workflow Manager parameters are set either to default values or to the values provided by the administrator. To change the values after installation, use the following information:

1. Open the Workflow Manager configuration file in a text editor:  
`$ACTIVATOR_ETC/config/mwfm.xml`

Table 5-1 lists the variable parameters in this file. The only required parameters are *Port* and *Max-Threads*

2. After editing the file and saving the changes, stop and restart the Workflow Manager or press reload configuration in the UI. The ‘Reconfigurable’ column indicates all parameters specified in the `mwfm.xml`, whose value can be changed during runtime. The initial parameters of all configured modules can also be changed during runtime. For more details on reconfigurable parameters of individual nodes, see “Using the Workflow Manager Module Library” on page 362.

For example, if a user wants to change the maximum length of the pending items that is expected to be run by one of the worker threads in the pool, the value of `Max-Work-List-Length` must be modified so that the configuration can be reloaded from the UI.

Similarly, if the interval at which the cluster node must update its heartbeat status needs to be increased or decreased, the initial parameter `keep_alive_time` of the `kee_alive` module must be modified so that the configuration can be reloaded from the UI.

**Table 5-1 Workflow Manager Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>Port</i>	Yes	The port that the Workflow Manager is bound to. The RMI remote object that you can interact with is exported into this port.	No	None
<i>Max-Threads</i>	Yes	The maximum number of threads that the Workflow Manager will use for its pool. This number limits the maximum number of process nodes being run at the same time.	Yes	None
<i>Min-Threads</i>	No	The minimum number of threads that the Workflow Manager will keep available to handle running workflows. If additional threads are required, the Workflow Manager creates dynamic threads up to the <code>Max-Threads</code> setting. (See the <code>Spawn_List_Length</code> discussion.) Dynamic threads expire after 10 seconds of inactivity, or the time specified in <code>Idle_Thread_Keep_Alive</code> .	Yes. Only increase is allowed, decrease will be ignored until next re-start	<code>Max-Threads</code>



**Table 5-1 Workflow Manager Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
<i>Idle_Thread_Keep_Alive</i>	No	The amount of time (in seconds) that an idle dynamic thread exists before it expires and is destroyed (see the <i>Min-Threads</i> discussion) . This only refers to dynamic threads.	Yes	<b>10</b>
<i>Max-Work-List-Length</i>	No	The maximum number of concurrently running jobs.	Yes	<b>512</b>
<i>Max-Nodes-Per-Thread</i>	No	The maximum number of workflow nodes for one job, which is executed in one worker thread before the worker thread is released. However the worker thread is always released when persistence is done.	Yes	<b>5</b>
<i>Test-Mode</i>	No	When set to true the workflow nodes which are marked with the test flag will be executed else they would not.	Yes	<b>false</b>
<i>Spawn-List-Length</i>	No	If the number of pending work items exceeds this number, the Workflow Manager spawns a new dynamic thread to handle them (assuming it has not exceeded <i>Max-Threads</i> ).	Yes	<i>1% of Max_Work_List_Length</i>
<i>Persistent-Timeouts</i>	No	This parameter tells the Workflow Manager whether or not to reset the time-outs when recovering after a shutdown. The time that the system has been down is also taken into account when calculating timeout expirations.	Yes	<b>“true”</b>
<i>Initial-Workflow-Load</i>	No	If true, on start-up the Workflow Manager attempts to read and validate all of the workflows in the <i>\$ACTIVATOR_ETC/workflows</i> directory. Any errors are logged to the <i>mwfm.stdout</i> log file.	No	<b>“true”</b>
<i>Statistics</i>	No	Specifies the maximum workflow runtime history for the statistics manager within a Service Activator workflow. If this parameter is absent, no statistics are performed. Using the statistic manager might slow down the system.	No	None
<i>Admin-Role</i>	No	Specifies the role that the Workflow Manager will consider to be an administrator. Some operations, such as the ability to delete all of the log files, are only authorized for users that are an administrator. In these cases, the Workflow Manager will test against this given role.	Yes	None

**Table 5-1 Workflow Manager Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
<i>Queue-Timeout-Seconds</i>	No	This parameter indicates how long a nonpermanent queue will exist after it becomes empty. The default is 0 (zero) seconds; if this parameter is set to 0 seconds, the Workflow Manager will demonstrate the same behavior that it did prior to the 4.0 release.	Yes	0
<i>Error-Messages</i>	No	You can set the queue name and role to be used for internally generated error messages. The default queue is "Errors." It is not created to be a permanent queue unless you declare it so. By default, the role associated with this queue is unassigned so anyone can view the error messages.  Use attributes on the <Error-Messages> tag to set these values, as shown here:  <Error-Messages queue="MyErrorQueue" role = "myrole" />	Yes	"Errors"
<i>Permanent-Queue</i>	No	Unless declared to be a permanent queue, the message and request queues that are created during workflow steps (like PutMessage and AskFor) will be removed from the system when they become empty (after the declared timeout). You can declare some queues to be permanent and indicate one or more roles that will be able to see the queue even if there is nothing in the queue for that role. If the type attribute is not specified on the Queue tag, both a "request" and a "message" queue will be created. See the examples following this table for additional information.	Yes	None
<i>Java-Class-Path</i>	No	Insert any extra jar or zip files to be searched by the java process and java rule nodes. The custom jar or zip files will be added to the head of the system generated list. Thus, any custom jar or zip files will be searched first when compiling/executing nodes. <Java-Class-Path>your path</Java-Class-Path>	No	None
<i>Dyn-Class-Path</i>	No	Path the dynamic classes created by the java rule and java process nodes	No	None
<i>Resource-Manager-RMI-Host</i>	No	The host name where the resource manger is running. Must be set to local host	No	None

**Table 5-1 Workflow Manager Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
<i>Resource-Manager-RMI-Port</i>	No	The RMI port for the Resource Manager.	No	None
<i>Resource-Manager-RMI-Name</i>	No	The name with which the resource manager RMI interface is bound.	No	None
<i>Management-Realm-User name</i>	Yes	The user name for the HTTP connection to the JBoss CLI	No	None
<i>Management-Realm-Password</i>	Yes	The password for the HTTP connection to the JBoss CLI	No	None

**Example 5-1 Creating a Permanent Message and Request Queues Viewable by “rolex” and “roley” Users**

```
<Permanent-Queue name="operator">
  <Role>rolex</Role>
  <Role>roley</Role>
</Permanent-Queue>
```

**Example 5-2 Creating a Permanent Message Queue Viewable by “admin” Users Only**

```
<Permanent-Queue name="admin" type="message">
  <Role>admin</Role>
</Permanent-Queue>
```

**Example 5-3 Creating a Permanent Request Queue Any User Can View**

```
<Permanent-Queue name="sync" type="request" />
```

## Understanding Workflow Manager Modules

You can configure the Workflow Manager to enhance or customize the behavior of the system using configurable modules. Use the built-in Workflow Manager modules, or write custom Workflow Manager modules to provide a new or enhanced service for the Workflow Manager.

You configure workflow modules in the file `$ACTIVATOR_ETC/config/mwfm.xml`. When the Workflow Manager starts, it reads `mwfm.xml` to determine which Workflow Manager modules to load and then loads these modules in the order that they are specified in the configuration file.

You configure each module by specifying a name for the module and a Java class that provides the implementation. Additionally, some workflow modules allow or require configuration parameters to specify their behavior. The name you give to the module is important because that is how the engine or other modules or workflow nodes can find the module they are interested in using. Each workflow module has a unique name in `mwfm.xml`. This is distinct from the name of the Java class that implements it.

Some modules require a specific name because the Workflow Manager looks for a module with that name. For example, to be able to do user authentication, there must be a module named “`authenticator`”. See the detailed discussion of each module for an indication of whether it must be given a specific name.

### Required and Typical Workflow Manager Modules

Some modules are required for Service Activator to function properly. In some cases, there is only one Java class that is available to provide the necessary functionality. In other cases, you have a choice of which Java classes to use to provide for the implementation.

#### Logging

The Workflow Manager looks for a module with the name “`log_manager`”. This module provides functions for logging operations. The “`XMLLogModule`” on page 448 is the built-in Java class to use for this functionality. The `XMLLogModule` logs messages using an XML format into regular files. The “`SolutionXMLLogModule`” makes it possible for a solution to generate its own log files.

#### Work Manager

The Workflow Manager processes the workflows one node at a time. When a workflow is started a work-item for processing the initial node of the workflow is placed on a work queue. The work queue is managed by a work manager module. As each node is completed, the thread processing the node places a new work-item for the next node in the workflow on the work queue, then the thread requests the next work item from the queue. Thus, the MWFM requires the presence of a work manager. If the `mwfm.xml` configuration file does not contain any module with the name “`work_manager`”, then the engine automatically uses the built-in `WorkManagerModule` introduced in release 3.6.

## Transactional State

Using a transaction module, the Workflow Manager maintains the state of running workflows in a persistent fashion in the database or the file system. This means that, if the Workflow Manager is shutdown and restarted or the workflows are taken over by another cluster node, any workflows that were running at the time of shutdown resume running at the place they left off when the manager restarts. This also safely handles the case where the Workflow Manager shuts down unexpectedly.

The Workflow Manager looks for a Workflow Manager module named “`transaction_manager`.” If it does not find a module with this name, it does not maintain a persistent state of running workflows.

When Running in a cluster environment you need to use the `DBTransactionModule` to ensure one cluster node can takeover jobs from another cluster node in case this node shut down unexpectedly.

If Service Activator is shut down while jobs are running and if the jobs must continue to run with the same workflow versions when restarted or when the jobs are taken over, a distribution module must be configured.

One class exists the “`DBTransactionModule`” on page 377 to use for this functionality.

## Activation

An activation module provides the interface between the Workflow Manager and the activation engine (Resource Manager). Virtually all installations of Service Activator has one modules configured with the names “`activator`”. The “`ActivationModule`” on page 363 is the built-in Java classes to use for this functionality.

## Authentication

The Workflow Manager looks for a module configured with the name “`authenticator`” and uses the module to answer three types of questions:

- Authentication - Can a user with this name and this password log in to the system?
- Authorization - Is this user in the proper role to perform this task?
- Valid roles - Which roles are considered valid in the system?

If it does not find the “`authenticator`” module, the Workflow Manager does not perform any user authentication. Without an authentication module, any user can interact with the system and perform any task. If no authenticator is configured, then the answer to the first two questions above is always “yes”. The valid roles will in this case be none.

The system comes with five Workflow Manager modules for authentication. They are:

- `DatabaseAdvancedAuthModule`, which authenticates by using information saved in the database through the User Management Interface (see “`DatabaseAdvancedAuthModule`” on page 374)
- `HPUXAdvancedAuthModule`, which authenticates HP-UX usernames (see “`HPUXAdvancedAuthModule`” on page 378).
- `LDAPAuthModule`, which authenticates usernames against an LDAP directory server (see “`LDAPAuthModule`” on page 394)
- `LinuxAdvancedAuthModule`, which authenticates Linux usernames (see “`LinuxAdvancedAuthModule`” on page 399)

- `WindowsAdvancedAuthModule`, which authenticates Windows usernames (see “`WindowsAdvancedAuthModule`” on page 444)

It is also possible to provide your own authentication module. See “Writing New Authenticator Module” on page 473.

### Database Access

The Workflow Manager looks for a module with the name “db”. This module will be used for all internal database access from the Workflow Manager. It is also this database which is often named “The System Database”.

One database module are provided with Service Activator the “`JNDIDatabaseModule`”. The module makes it possible to provide access to one or more databases that have a JDBC driver. It is typical to have a single module configured. Many of the built-in workflow nodes, by default, look for a module with the name “db” and a number of nodes would always use the module configured with the name “db”.

### Parent-Child Synchronization

The Workflow Manager looks for a module with the name “sync\_module”. Sync module will be used to synchronize parent and child workflows. The “`SyncModule`” on page 436 is the only built-in Java class to use for this functionality. Every installation of Service Activator will have this module configured automatically. This will be used in both standalone and distributed mode of operation. The Workflow Manager requires the presence of a Sync module.

### Receiving Messages

Typical configuration of the Workflow Manager have a module that receives messages and launches workflows to process each message. These messages arrive through a communication mechanism, such as a socket or bus.

The simplest communication mechanism is via sockets. The `SocketListenerModule` receives messages on a waiting socket and launches workflows to process each arriving message.

This functionality can be provided by a variety of Java classes, each using a different communication mechanism. The only module provided with Service Activator is the `SocketListenerModule`.

Because no other component of the system needs to obtain a handle to the `SocketListenerModule`, there are no other requirements for the name given to the module except that the name must be unique.

### Sending Messages

Typical configurations of the Workflow Manager have a module to send messages from workflows to acknowledge the status of completed tasks. Here again is functionality that could be provided by a variety of Java classes.

The only module provided with Service Activator for this purpose is the `SocketSenderModule`. The `SocketSenderModule` sends messages from workflows to processes listening on a socket port.

The name given to the `SocketSenderModule` is important because your workflows (in the `SendMessage` nodes) need to refer to the module by its name. Although there is not any default or recommended name, “sender” is a useful convention to adopt.

## Keep Alive

The keep alive module handles failover of jobs and monitoring of other cluster nodes in a distributed environment. Apart from this it is also monitoring the Resource Manager and the database connectivity to the System Database. This enables to overcome issues like machine failure or loss of database connectivity. This means that, if the workflow manager running in a cluster node is shutdown or a machine fails, the jobs running in that cluster node are continuously failed over to another cluster node.

The Workflow Manager must be configured with a module named “keep\_alive.”

The “KeepAliveModule” on page 390 is the only built-in Java class to use for this functionality.

## Distribution

Using a distribution module, the Workflow Manager handles load balancing in a distributed environment. This enables to distribute workflow execution to other nodes in a cluster. In case of an standalone environment, it would distribute the jobs by itself.

If Service Activator is running with a distribution module configured and a request to start a job is received by the Workflow Manager, it requests the distribution module to handle the load balancing.

The jobs are distributed among the currently active cluster nodes, which are online, not suspended, and not locked. If none of the nodes are active, an attempt will be made to start the job on the node which initiates the distribution. This is necessary in cases where a parent workflow starts a child workflow, and it should be permitted even if the cluster node is in a locked state.

A number of different distribution modules exists and the only difference between them are the algorithm used for load balancing. For a particular instance, only a single distribution module can be configured which will be used for distribution. All cluster nodes in a Service Activator installation must be running with the same distribution module.

The distribution module must be configured with the name “distribution\_module”

The system comes with the following three load balancing schemes:

- RoundRobinDistModule, which distributes jobs across the cluster nodes in a round robin fashion. ( see “RoundRobinDistModule” on page 421)
- LoadFactorDistModule, which distributes jobs across the cluster nodes based on the load factor. ( see “LoadFactorDistModule” on page 401)
- QueueDistModule, which distributes jobs across the cluster nodes based on the number of currently running jobs. (see “QueueDistModule” on page 416)





---

## **6 Workflow Manager Module Library**

The Workflow Manager comes with an extensive library of workflow modules. Each supplied module is described in detail here. Specific instructions for configuring each module are included.

## Using the Workflow Manager Module Library

This chapter describes each of the built-in modules that you can use to configure the Workflow Manager. While it is always possible to write your own Java classes, in most cases these built-in modules provide all the functionality you need.

These modules are configured in the file `$ACTIVATOR_ETC/config/mwfm.xml`. The examples shown for each module are the XML that would be placed into this file.

## ActivationModule

`com.hp.ov.activator.mwfm.engine.module.ActivationModule`

The `ActivationModule` links the Workflow Manager with the default activation engine (Resource Manager).

The module is accessed by the `Activate` workflow node. By default, the `Activate` node uses a module with the name “`activator`”. The `Activate` node may be configured to use a module with a different name.

If you do not use the default activation engine (`ResourceManager`), or if you do not use the `Activate` node, make sure this module is removed from the `mwfm.xml` file.

When using this module, the `Activate` node places an activation request on an activation queue for processing by the activation threads that are managed by this module. In addition, the node behaves like the `AskFor` node in that it posts a request on one of the request queues (named “`activation`”), thus, freeing the workflow thread for processing other workflows. The request is placed with the role “`internal`”. Thus, normal users do not see these requests unless they have the role “`internal`”. When activation completes, the module sends a response to the waiting request.

The activation threads also support prioritized handling of items in the activation queue. When an activation request is placed on the activation queue, the case-packet of the workflow is examined. If the case-packet contains a variable with the name `PRIORITY`, the value of it is used to order the processing of items in the queue. Items of a higher priority value are processed before items of a lower priority value. If the `PRIORITY` variable is not found, a priority of 0 is assumed. It is possible to assign a negative priority. The value of the `PRIORITY` variable is pass to the Resource Manager which then uses this value when an atomic task is finished with its use of a lock and another atomic task can be started.

The module is hardcoded to put all activations into the `Activation` queue. The job counter in the Operator UI uses this queue to calculate the number of outstanding activations.

### See Also

- “`Activate`” on page 96 in this guide
- “`Job Counters`” on page 96 in *HP Service Activator - Introduction and Overview*

**Table 6-1**      **ActivationModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<code>min_threads</code>	No	The minimum number of threads to maintain for executing activations.	Yes. Only increase is allowed, decrease will be ignored until next re-start.	5
<code>max_threads</code>	No	The maximum number of threads to allow for executing activations.	Yes	20

**Table 6-1      ActivationModule Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
<i>queue_class</i>	No	<p>This can be set to the <code>com.hp.ov.activator.mwfm.module.WeightedEngineQueue</code>, <code>com.hp.ov.activator.mwfm.module.SimpleEngineQueue</code>, or <code>com.hp.ov.activator.mwfm.module.PriorityEngineQueue</code></p> <p>The <code>WeightedEngineQueue</code> use the <code>PRIORITY</code> case-packet variable in a weighted way to prioritize the items on which the activation threads operate. Items that have the same priority will be processed in FIFO order.</p> <p>The <code>SimpleEngineQueue</code> will not do any prioritization of activation requests. They will be processed in FIFO order.</p> <p>The <code>PriorityEngineQueue</code> uses the <code>PRIORITY</code> case-packet variable to prioritize the items on which the activation threads operate. Items that have the same priority value will be processed in FIFO order.</p>	No	<code>com.hp.ov.activator.mwfm.module.WeightedEngineQueue</code>
<i>refresh_interval</i>	No	The time interval between refreshing the internal cache which keeps data about which atomic tasks are deployed.	No	60000 milliseconds

**See Also**

- “Activate” on page 96 for more information about performing an activation from a workflow.

**Example 6-1      ActivationModule**

```

<Module>
  <Name>activator</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.ActivationModule
  </Class-Name>
  <Param name="min_threads" value="5"/>
  <Param name="max_threads" value="20"/>
</Module>

```

---

## AuditModule

`com.hp.ov.activator.mwfm.engine.module.DBAuditModule`

This workflow module provides the auditing mechanism that writes audit records using system database module. The `AuditModule` is used by the `Audit` node to write audit records and by the statistical servlet to collect workflow statistics. The module is also used to write an audit record when starting or killing a job and for erroneous workflow ends.

When the `Audit` node writes an audit record, the event type of the audit record will be “`LOG_EVENT`” by default. The event type used by the `Audit` node can be changed to any other event type (a `String`). Note that you can not use some reserved event types are these:

- `KILL_JOB_EVENT`
- `KILL_JOB_NULL_USER_EVENT`
- `START_JOB_EVENT`
- `END_JOB_EVENT`
- `EXCEPTIONAL_JOB_EVENT`
- `START_JOB_NULL_USER_EVENT`
- `INVENTORY_EVENT`

The default audit module used by the `Audit` node is “`auditor`”. This can also be changed.

When a job is started interactively (command line or operator UI), the event type is `START_JOB_EVENT`. If the start job occurs via another workflow or the `SocketListenerModule`, the event type is `START_JOB_NULL_USER_EVENT`.

There are four different events for workflow end.

- `END_JOB_EVENT`. This event occurs when workflows finish their jobs gratefully.
- `KILL_JOB_EVENT`. This event occurs when a job is killed interactively (command line or the Operator UI).
- `KILL_JOB_NULL_USER_EVENT`. This event occurs when a job is killed via another workflow.
- `EXCEPTIONAL_END_JOB_EVENT`. This event occurs if a workflow is erroneous.

If an audit module with the name “`auditor`” is not specified, then auditing or collecting workflow statistics is not performed. You can enable or disable workflow statistics by setting the `store_statistics` parameter to “`true`” or “`false`”. Note that you can also enable or disable auditing or collecting workflow statistics for a particular (individual) workflow. To do this, change the workflow parameters in the Workflow Designer. Remember that workflow statistics are collected for `END_JOB_EVENT` only, which means that workflow statistics is not available for erroneous or killed workflows.

Information about which fields are written in an audit record can be found in `$ACTIVATOR_ETC/sql/createAuditDB.sql`.

An audit record consists of one or more records in the database. It is possible to associate additional data to an audit record using the audit parameters. In the default setup, the value of a parameter is a string of max 200 characters. However you can modify the value to be of type LONG meaning that up to 2 GB of data can be stored. The change is only possible specifically for the Value column.

Use the following command to alter the column type:

```
alter table audit_record_params modify (value LONG)
```

To change the column type to the default, type:

```
alter table audit_record_params modify (value VARCHAR2(200))
```

Audit records are always saved in the System Database. The database module configured with the name “db”.

**Table 6-2 AuditModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>exclude0</i> <i>exclude1...</i> <i>excludeN</i>	No	Event types to exclude from auditing:  KILL_JOB_EVENT KILL_JOB_NULL_USER_EVENT LOG_EVENT START_JOB_EVENT START_JOB_NULL_USER_EVENT END_JOB_EVENT EXCEPTIONAL_END_JOB_EVENT  Your custom event type can also be excluded by specifying it in this list.	Yes. You can add or remove as many <i>excludeX</i> parameters as you need	None
<i>store_statistics</i>	No	Indicates whether to store or not workflow statistics.	Yes	“false”
<i>store_audit</i>	No	Indicates whether to store or not audit records for all workflows.	Yes	“false”

---

## BusinessCalendarModule

```
com.hp.ov.activator.mwfm.engine.module.umm.BusinessCalendarModule
```

The module enables workflows to work with the business calendars defined in Service Activator. Workflows use the nodes specially developed for the business calendar to retrieve and calculate business hours information.

The business calendar module retrieves the calendar definition from the database and stores the information in an in-memory cache, facilitating the use of the business calendar data by the workflow nodes. The business calendar module also serves as the interface for the Web UI to update calendar information such as business hours, holidays and recurring holidays.

This module supports reconfiguration. Reloading the configuration will re-initialize the in-memory cache.

The following is the configuration for the business calendar module. The name of the module must "business\_calendar\_module".

### Example 6-2

#### BusinessCalendarModule

This example configures the BusinessCalendarModule.

```
<Module>
  <Name>business_calendar_module</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.BusinessCalendarModule
  </Class-Name>
</Module>
```

---

## CacheModule

`com.hp.ov.activator.mwfm.engine.module.CacheModule`

The module provides a cache for inventory database queries. The module works with the QueryInventory node. The module can be configured more than once with different names. The name is also specified on the QueryInventory node to bind the node to the module instance.

**Table 6-3** CacheModule Parameters

Parameter	Required	Description	Reconfigurable	Default
<i>timeout</i>	Yes	Default timeout used when no value is provided by the node using this module. The value is specified in seconds.	No	None
<i>check_interval</i>	Yes	Time, in seconds, between every check for expired content.	No	None

**Example 6-3** CacheModule

This example configures the CacheModule.

```
<Module>
  <Name>check_time</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.CacheModule
  </Class-Name>
  <Param name="timeout" value="60"/>
  <Param name="check_interval" value="10"/>
</Module>
```



## CasePacketDistModule

`com.hp.ov.activator.mwfm.engine.module.umm.CasePacketDistModule`

This module allows the workflow manager to perform load balancing of workflow execution based on the value of a case-packet.

Load balancing can be switched off using the parameter "dispatch\_local". The job is always executed in the local host.

The parameter "casepacket" specifies the case-packet that decides the load balancing. The possible values for this case-packet can be specified using the parameters value0, value1...valueN. The cluster nodes where the workflow can be processed are specified using the parameters hostname0, hostname1...hostnameN.

The number of possible values and possible hostnames specified must be the same.

### Request to start a job with a set of initial case-packets

When a request to start a job is received then a check is made if the initial case-packets that accompany the request contain the case-packet.

If the case-packet is found then its value is matched with the configured values. If a match is found then the workflow is dispatched to the corresponding host.

### Request to start a job without a set of initial case-packets

If a request to start a job is received without any initial case-packets then job is dispatched to the default host. If the default host is not specified then its dispatched to the local host.

**Table 6-4 CasePacketDistModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>dispatch_loca l</i>	No	Configurable value which decides if load balancing is performed or not. If this parameter is set to true then load balancing is switched off and jobs are dispatched only to the local cluster node	Yes.	False
<i>casepacket</i>	Yes	The name of a case-packet variable which decides the distribution	Yes	None
Value0, Value1 ... Value N	No	The possible values for the parameter <i>casepacket</i>	Yes	None
hostname0, hostname1 ... hostnameN	No	The cluster nodes where the job can be distributed	Yes	None

**Table 6-4 CasePacketDistModule Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
default	No	The cluster node where the job will be distributed if the value of the specified case-packet does not match any of the values specified by value0, value1...valueN	Yes	None

**Example 6-4 CasePacketDistModule**

If configuration is as follows:

```

<Module>
  <Name>distribution_module</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.umm.CasePacketDistModule
  </Class-Name>
  <Param name="dispatch_local" value="false"/>
  <Param name="casepacket" value="casepacket1"/>
  <Param name="value0" value="value1"/>
  <Param name="value1" value="value2"/>
  <Param name="value2" value="value3"/>
  <Param name="hostname0" value="host1"/>
  <Param name="hostname1" value="host2"/>
  <Param name="hostname2" value="host3"/>
  <Param name="default" value="host3"/>
</Module>

```

If the value of the case-packet "casepacket1" in the initial case-packets is value1 then the job is dispatched to host1. The cluster node "host1" must be active.

If a match is not found then the job is dispatched to the host specified by the configuration parameter "default". The default host is optional. In this case the request is dispatched to the local host.

If the case-packet "casepacket1" is not found in the initial case-packets then the job is dispatched to the default host.

If the start of a job fails if the matching cluster node gets suspended or locked or its being shutdown or its max job limit has reached between the times when request is dispatched and when it reaches the cluster node, an attempt is made to dispatch the request to the default host.

## CheckTimeModule

`com.hp.ov.activator.mwfm.engine.module.CheckTimeModule`

The module checks if the system time on all cluster nodes are the same. With a given configurable interval the module will ask all the cluster nodes for their system time and if the time returned compared with the local system time is greater than the configurable time difference an ERROR message will be written in the log file.

This module should only be used when Service Activator is configured with more than one cluster node. The module can be started both when Service Activator is started or as part of a reconfiguration. The module uses the master slave approach so only one cluster node, the master, will perform requests to the other cluster nodes.

**Table 6-5 CheckTimeModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>check_time_poll_interval</i>	No	Time interval in milliseconds at which this module will periodically ask each cluster node to give its system time	Yes.	10000
<i>allowable_time_delta</i>	No	Configurable value in milliseconds which is the allowable difference between the system times on master node and slave node.	Yes	20000

**Example 6-5 CheckTimeModule**

This example configures the CheckTimeModule.

```
<Module>
  <Name>check_time</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.umm.CheckTimeModule
  </Class-Name>
  <Param name="check_time_poll_interval" value="10000"/>
  <Param name="allowable_time_delta" value="20000"/>
</Module>
```



## ConflictModule

`com.hp.ov.activator.mwfm.engine.module.ConflictModule`

This module must be configured when some of the mutex workflow nodes are used. The module controls the mutexes which are locked by the different jobs and grant a job a mutex when it is released by another job.

### See Also

- “MutexLock” on page 219
- “MutexUnlock” on page 222
- “MutexGetInfo” on page 218
- “MutexSetInfo” on page 221

**Table 6-6** ConflictModule Parameters

Parameter	Required	Description	Reconfigurable	Default
<i>poll_interval</i>	No	This parameter controls the time the conflict thread will check for if a retry should be done to acquire a mutex.	No	10000 ms

**Example 6-6** ConflictModule

```
<Module>
  <Name>conflict_module</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.ConflictModule
  </Class-Name>
  <Param name="poll_interval" value="10000"/>
</Module>
```

## DatabaseAdvancedAuthModule

`com.hp.ov.activator.mwfm.engine.module.umm.DatabaseAdvancedAuthModule`

This is Service Activator's native module for authentication of users and authorization to use roles. It is independent of the host operating system and uses data that is maintained by the user management functions available to system administrators and stored in the system database

This module is not configured at installation time. To use it, configure it with name `authenticator`.

During installation of Service Activator the system user and the roles “admin” and “internal” are created. The system user login account must be used to create other users and roles.

### See Also

- “Roles, Privileges, and Authentication” in the *HP Service Activator - System Integrator's Overview*
- “User Management” in *HP Service Activator, HP Service Activator - User's and Administrator's Guide*
- “Configuring Authentication or Authorization” on page 68 in *HP Service Activator—Developing Plug-Ins and Compound Tasks*

**Table 6-7**      **DatabasedvancedAuthModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<code>mwfm_remote_url</code>	Yes	URL where the module will access the RMI of the Workflow Manager. When the default port is used (you can see the port at the beginning of the <code>mwfm.xml</code> file) the URL is:  <code>//localhost:2000/wfm</code>	No	None
<code>sleep_time</code>	No	The time between the internal role cache is cleared.	No	30 min
<code>eight_char_password</code>	No	If password should be truncated to 8 characters before authentication is done	No	false
<code>secure_username</code>	No	Transform the user name to a valid value, cutting the user name from the first invalid character.	No	false
<code>password_validation</code>	No	true/false: specifies if password validation and expiration shall apply	No	false

**Table 6-7 DatabasedvancedAuthModule Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
expiry_days	No	Number of days from a password is assigned until it expires. 0 means never.	No	0
expiry_alert_days	No	Starting a number of days before a password expires, the user is warned at login. This parameter specifies the number.	No	0
reuse_interval	No	The number of distinct values that must be assigned to a password over time before one can be reused.	No	0
teams_enabled	No	Name of the parameter which indicates whether teams will be used or not.	No	false
format_checker_class	No	<p>The format checker class, where the complexity of the password is analyzed before accepting it.</p> <p>HPSA provides a SimplePasswordFormatChecker which checks that the password is not equal to the user name and a ComplexPasswordFormatChecker which also checks the password strength: usage of numbers and case letters.</p> <p>Any other class may be defined here. If so, you can implement your password checker class just by extending the <code>com.hp.ov.activator.mwfm.engine.module.umm.pwd.PasswordFormatChecker</code>.</p>	No	<code>com.hp.ov.activator.mwfm.module.umm.SimplePasswordFormatChecker</code>
allowed_invalid_login_attempts	No	The number of previous consecutive invalid login attempts allowed for each user before he becomes disabled. A value of 0 means that users will never be disabled.	No	0

**Example 6-7 DatabaseAdvancedAuthModule**

This example configures the DatabaseAdvancedAuthModule with the workflow Manager service.

```
<Module
  <Name>authenticator</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.umm.DatabaseAdvancedAuthModule
  </Class-Name>
  <Param name="mwfm_remote_url" value="//localhost:2000/wfm" />
</Module>
```



---

## DBTransactionModule

```
com.hp.ov.activator.mwfm.engine.module.DBTransactionModule
```

This module allows the Workflow Manager to handle persistence and retrieval of running jobs information in the system database. This enables jobs to survive machine failures in both distributed and standalone environment. In case of server failure, if the module is not specified, case packets and process states are lost. Make sure to specify the module name as "transaction\_manager".

This module depends on the OracleDatabaseModule to obtain database connections. The database module which is used is the one named db.

If the Workflow Manager is shutdown while jobs are running, the module retrieves all the case-packet states from the database on restart and allows the jobs to be restarted from the same state.

When a job is started, it would use the current active workflow saved in the database and even if you reload a newer version of the workflow, the job will continue to use the one which it was started with. This would also be the situation if service activator is restarted or the job fails over on an other cluster node.

On completion of a job, the module also deletes the case-packet from the database and the workflow in case no other jobs are referring to this workflow and the workflow is not the newest version of the workflow.

The module works in conjunction with the KeepAlive module in a failover scenario to retrieve case-packets of jobs which were being executed by the failed node.

### See Also

- “JNDIDatabaseModule” on page 389
- “KeepAliveModule” on page 390

### Example 6-8

#### DBTransactionModule

```
<Module>
  <Name>transaction_manager</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.DBTransactionModule
  </Class-Name>
</Module>
```

---

## HPUXAdvancedAuthModule

`com.hp.ov.activator.mwfm.engine.module.umm.HPUXAdvancedAuthModule`

This module provides authentication and authorization functionality based on the underlying HP-UX Operating System authentication mechanism. It is only suitable in a HP-UX installation.

The roles defined in the operating system and used by Service Activator must also be created through the User Management Interface. Only users which have this configuration are able to login to Service Activator. During installation of Service Activator the roles “admin” and “internal” are created for User Management. The “admin” role must be configured in the operating system to make it possible to enable the HPUXAdvancedAuthModule. However it is possible to use role mapping and hence create a different role for the System User in the underlying operating system.

Name this module “authenticator” in the configuration.

### See Also

- “Roles, Privileges, and Authentication” in the *HP Service Activator - System Integrator’s Overview*
- “Configuring Authentication or Authorization” on page 68 in *HP Service Activator—Developing Plug-Ins and Compound Tasks*

**Table 6-8** HPUXAdvancedAuthModule Parameters

Parameter	Required	Description	Reconfigurable	Default
<code>mwfm_remote_url</code>	Yes	Used to indicate from where to get the remote Workflow Manager service	No	None
<code>sleep_time</code>	No	The time between the internal role cache is cleared.	No	30 min
<code>eight_char_password</code>	No	If password should be truncated to 8 characters before authentication is done	No	false
<code>secure_username</code>	No	Transform the user name to a valid value, cutting the user name from the first invalid character.	No	false

**Example 6-9**      **HPUXAdvancedAuthModule**

This example configures the HPUXAdvancedAuthModule with the workflow Manager service.

```
<Module
  <Name>authenticator</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.umm.HPUXAdvancedAuthModule
  </Class-Name>
  <Param name="mwfm_remote_url" value="//localhost:2000/wfm" />
</Module>
```

## HTTPRenderer

```
com.hp.ov.activator.mwfm.engine.module.monitor.HTTPRenderer
```

Use this module along with `Monitor`, described on page 407, to render (in HTML) data accumulated about the functioning of the Workflow Manager. Name this module “HTTPRenderer”.

Once configured, two different URLs become available:

- `http://yourhost:renderer_port/server`  
Shows statistics about the Workflow Manager performance.
- `http://yourhost:renderer_port/statistics`  
Shows statistics about the different workflows that have been run, taking into account the history size specified in the `statistic` parameter of the Workflow Manager configuration.

Specify the TCP port for the module to listen on to receive such information (the default is 7070).

The module does not support re-configuration.

The module should only be used during development as the module is very memory consuming. The module must not be used in production.

### Example 6-10

#### HTTPRenderer

```
<Module>  
  <Name>HTTPRenderer</Name>  
  <Class-Name>  
    com.hp.ov.activator.mwfm.engine.module.monitor.HTTPRenderer  
  </Class-Name>  
</Module>
```

---

#### NOTE

Because the module consumes a lot of memory, it is advisable not to use this in a production environment.

---

## HTTPSenderModule

`com.hp.ov.activator.mwfm.engine.module.monitor.HTTPSenderModule`

The HTTPSenderModule provides a mechanism for the workflows to make HTTP(S) GET or POST requests.

The module opens an http(s) connection to target URL. The module can then be accessed by HTTPRequest workflow node to make an asynchronous GET or POST. You can configure multiple HTTPSenderModule; each must be given a unique name. The HTTPRequest can be configured to use any one of the modules using its name.

The module also supports the following additional features.

- HTTPS Server/Client side certificates
- Proxy server
- HTTP basic username/password authentication for network connection
- Customizable timeout value

The HTTPSenderModule ensures that multiple http requests are processed and incoming requests are not blocked while existing requests are being processed. When using this module, it places a request by the HTTPRequest node in a queue for processing by the http listener threads that are managed by this module. In addition, the node behaves like the AskFor node in that it posts a request on one of the request queues (named "httprequest"), thus, freeing the workflow thread for processing other workflows. The request is placed with the role "internal". Thus, normal users do not see these requests unless they have the role "internal". After processing the http request, the module sends the response and any returned cookie to the waiting request.

The only necessary items to configure is the for the HTTP(S) connection.

You can configure this module to make a normal http connection or a secure http connection to the server. In order to make an https connection you must provide a valid SSL certificate identifying the server.

---

### NOTE

Keystore, storepass, and keypass must all be set to some non-empty values for the module to do HTTPS client authentication; otherwise there will be no effect

For network connection authentication the parameters `username` and `password` must be specified.

If the target URL must be accessed through a proxy server then the `proxy_server` and `proxy_port` must be specified.

---

### NOTE

Note that the parameters `proxy_server` and `proxy_port` must both be set to some non-empty values in order for the module to set up the proxy connection. Setting only one of them will not have any effect.

Connection and read timeouts can be specified using parameters `connect_timeout` and `read_timeout`.

The module does not support re-configuration.

**Table 6-9 HTTPSenderModule Parameters**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Reconfigurable</b>	<b>Default</b>
<i>host</i>	Yes	The target URL for the HTTP(S) connection	No	None
<i>keystore</i>	No	The location of the SSL keystore file necessary for HTTPS client authentication	No	30 min
<i>keypass</i>	No	The password for the public certificate/private key pair, necessary for HTTPS client authentication	No	None
<i>storepass</i>	No	The password to access the keystore file, necessary for HTTPS	No	None
<i>username</i>	No	Username for network connection authentication	No	None
<i>password</i>	No	Password for network connection authentication	No	None
<i>proxy_server</i>	No	Name of a proxy server if proxy is to be used	No	None
<i>connection_timeout</i>	No	http(s) connection timeout value, in milliseconds	No	None
<i>password</i>	No	Password for network connection authentication	No	None
<i>read_timeout</i>	No	Read timeout value, in milliseconds	No	None
<i>min_threads</i>	No	The minimum number of threads created to process http requests	No	1
<i>max_threads</i>	No	The maximum number of threads created to process http requests. This is the number of simultaneous requests that can be processed. Other incoming requests will be queued until one of the threads becomes available	No	3

---

## JMSListenerModule

`com.hp.ov.activator.mwfm.engine.module.JMSListenerModule`

The module connects to a JMS destination supported by a JMS Provider and waits for incoming messages. When a message arrives it is saved in a row of a database table that is used for temporary data (`database_message`), and a workflow job is started to process the message. A handle (URL) for the workflow job to retrieve the message is passed through the case-packet variable `message_url`. The workflow is responsible for cleaning up the database by removing the entry when it has been processed (using the `RemoveData` node).

The JMS destination can be a queue or a topic. The module must be configured with information to bind itself to the desired JMS destination by looking it up through a JNDI service. For a brief introduction to these JMS concepts refer to the appendix "*Java Message Service*" in *Service Activator, System Integrator's Overview*.

Configure as many instances of the `JMSListenerModules` as necessary. Each listener module must be given a unique name and destination to listen to and may start a unique workflow.

When *Service Activator* runs on a cluster of servers, the module should, like other workflow manager modules, be configured identically on all the cluster nodes. The JMS listener module will then only be active on one of the cluster nodes, the so-called master node; the nodes will not compete to extract received messages from the JMS provider. If the master node fails, a new master will be designated and its JMS listener module will become active. This behaviour is automatic and not configurable for the module.

The JMS Listener prefixes the value of the "jms\_destination" parameter with "topic/" and "queue/" when running in "topic" and "queue" mode, respectively.

The module is multi-threaded and avoids blocking behavior.

By default the module will prefix an XML header to the beginning of each received message. The header will normally refer to a DTD file. It will look like this:

```
<?xml version="1.0" encoding="utf-8"/>
<!DOCTYPE msg SYSTEM
"file://etc/opt/OV/ServiceActivator/config/exchange.dtd">
```

The name of the DTD file and the root tag can be configured. This behaviour can also be disabled.

---

### NOTE

Messages received by the `JMSListenerModule` must be in UTF-8 format.

### See Also

- "Java Message Service" in the *HP Service Activator - System Integrator's Overview*

**Table 6-10 JMSListenerModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>workflow</i>	Yes	Name of the workflow that is run to process each received message.	Yes	None
<i>jndi_initial_context_factory</i>	Yes	The JNDI initial context factory	No	None
<i>jndi_url_package_prefixes</i>	No	package prefixes to use when loading in URL context factories. For JBossMQ it is org.jboss.naming:org.jnp.interfaces	No	None
<i>jndi_url</i>	Yes	The URL to reach the JNDI service to lookup the JMS destination. The format will <url prefix>://<ip>:<port>	No	None
<i>connection_factory_name</i>	Yes	JMS connection factory name	No	None
<i>jms_transfer_mode</i>	Yes	JMS transfer mode, "topic" or "queue"	No	None
<i>jms_destination</i>	No	Name of the destination to listen at (queue or topic)	No	None
<i>durable</i>	No	Indicates if a durable subscription shall be created	No	false
<i>client_id</i>	Yes if durable	Client id for durable topic subscription	No	None
<i>unsubscribe</i>	No	Specifies whether the durable subscription must be unsubscribed when the module is stopped	Yes	false
<i>username</i>	Yes	User identity to create JMS connection	No	None
<i>password</i>	Yes	Password to create JMS connection	No	None
<i>retry_interval</i>	No	Interval to wait between attempts to connect to the JMS provider. Defined in milliseconds	No	1000



**Table 6-10 JMSListenerModule Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
<i>max_retry</i>	No	Number of retries to connect to the JMS provider before giving up. If set to 0 the module will continue to try to reconnect.	No	3
<i>recover</i>	No	When the module is started, if this parameter is true, all unprocessed messages (i.e. received into the database and not cleared by the workflow) will cause a workflow job to be started. This may be useful only if the Workflow Manager is not configured to persist the state of running workflows.	No	false
<i>header</i>	No	Specifies whether or not to put an XML header before each received message. Possible values are true and false.	No	true
<i>dtd_root_tag</i>	No	If a DTD is present (set with dtd parameter), the value of this parameter value will be put in each message header to identify the root of the message.	No	msg
<i>dtd</i>	No	Specifies the document type definition for received messages. The DTD can be used by the workflow to validate the XML syntax of the message. You can specify the absolute location of the DTD file, or a path relative to \$ACTIVATOR_ETC/config. To disable placing the DTD file name in the message header, set the value to NO_DTD.	No	Exchange.dtd
<i>min_threads</i>	No	The minimum number of threads to use for processing arriving messages.	No	1

**Table 6-10 JMSListenerModule Parameters (Continued)**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Reconfigurable</b>	<b>Default</b>
<i>max_threads</i>	No	The maximum number of threads to use for processing arriving messages.	No	1
<i>max_queue_length</i>	No	The maximum number of work items that can be present in the internal queue.  Work Items are added to this queue as and when messages are received from the JMS provider.  Only if the number of work items is below this value we will receive and process a new message.	Yes	50

## JMSSenderModule

`com.hp.ov.activator.mwfm.engine.module.JMSSenderModule`

The module works with the SendMessage workflow node to send messages to external systems via a JMS provider.

The module connects to a JMS destination, which can be a queue or a topic. It must be configured with information to bind itself to the desired JMS destination by looking it up through a JNDI service. For a brief introduction to these JMS concepts refer to the appendix "Java Message Service" in *Service Activator, System Integrator's Overview*.

Configure as many instances of the JMSSenderModules as necessary. Each sender module must be given a unique name and may send to a different destination.

### See Also

- "Java Message Service" in the *HP Service Activator - System Integrator's Overview*

**Table 6-11 JMSSenderModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>jndi_initial_context_factory</i>	Yes	The JNDI initial context factory	No	None
<i>jndi_url_package_prefixes</i>	No	package prefixes to use when loading in URL context factories. For JBossMQ it is org.jboss.naming:org.jnp.interfaces	No	None
<i>jndi_url</i>	Yes	The URL to reach the JNDI service to lookup the JMS destination. The format will <url prefix>://<ip>:<port>	No	None
<i>connection_factory_name</i>	Yes	JMS connection factory name	No	None
<i>jms_transfer_mode</i>	Yes	JMS transfer mode, "topic" or "queue"	No	None
<i>jms_destination</i>	No	Name of the destination to listen at (queue or topic)	No	None
<i>username</i>	Yes	User identity to create JMS connection	No	None
<i>password</i>	Yes	Password to create JMS connection	No	None

**Table 6-11 JMSSenderModule Parameters (Continued)**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Reconfigurable</b>	<b>Default</b>
<i>retry_interval</i>	No	Interval to wait between attempts to connect to the JMS provider. Defined in milliseconds	No	1000
<i>max_retry</i>	No	Number of retries to connect to the JMS provider before giving up	No	3

## JNDIDatabaseModule

`com.hp.ov.activator.mwfm.engine.module.JNDIDatabaseModule`

The module provides access to a relational database using a datasource configured in JBoss. The `JNDIDatabaseModule` is the default and preferred database module.

You can configure as many database modules as necessary to provide access to one or more databases. Typically, you configure only a single database module named “db” which is also the system database.

If you choose to configure multiple database modules, give each module a unique name.

During installation/configuration of Service Activator, one module is automatically configured. If you change the database then you will need to manually update the datasource parameters. Also the datasource which is configured in the module is created during installation/configuration. The datasource is configured in the file `$JBOSS_HOME/standalone/configuration/standalone.xml`.

**NOTE**

The password used to establish the database connection is in clear-text here. To protect this password, be sure that the `mwfm.xml` file is readable only by users with the appropriate privileges.

**Table 6-12 JNDIDatabaseModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<code>datasource_name</code>	Yes	The name of the datasource in JBoss which is used	No	None

**Example 6-11 JNDIDatabaseModule**

```
<Module>
  <Name>db</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.JNDIDatabaseModule
  </Class-Name>
  <Param name="datasource_name" value="java:/hpsa/jdbc/mwfmDB"/>
</Module>
```

## KeepAliveModule

`com.hp.ov.activator.mwfm.engine.module.KeepAliveModule`

The Workflow Manager requires a keep alive to perform the following tasks:

- Monitor the status of other nodes in a cluster and update its active status at regular intervals.
- Monitor the database connectivity of workflow manager and the resource manager, and also suspend the system in case of loss of database connectivity.
- In case of a cluster node failure, perform failover of jobs continuously by using the distribution module to distribute jobs among active cluster nodes.
- Update the running modules and the resource manager whenever a cluster node is down or starts up.
- Set and maintain administrative state of a cluster node or all cluster nodes to lock or unlock and update all modules and the resource manager when a cluster node is locked or unlocked. When a cluster node is locked, all running workflows on the node will continue to run, but all requests to start new workflows will be rejected.
- Set and maintain operational status of a cluster node or all cluster nodes to suspended or resumed and update all modules and the resource manager when a cluster node is suspended or resumed. When a cluster node is suspended, all activities are stopped, but the state of each job is maintained and all requests to start new workflows are rejected.

This feature is implemented by having a specific table in the database a clusternode list with one row for each node which must be updated. For example, every second (configurable) for each cluster node by a specific keep alive thread. If the row is not updated after a configurable amount (longer than twice the `keep_alive_time`) of time, other cluster nodes will start to takeover the work that must be done- run workflows, rollback of the transactions, and release of the locks in the resource manager. The `take_over_time` parameter determines as to how long a node can continue to be down before another cluster node takes over the work.

The system database is used for persistent of lot of different data then if it is not possible to obtain a connection to the database all processing is suspended and the database is polled for connectivity. If a connection is established, jobs are resumed again but only after ensuring that no other Workflow Manager has taken over the work.

The normal functioning of KeepAlive module can be turned off by setting the `update_heartbeat` parameter to false. This will be usually used when operating in a standalone environment. In this mode, only the database connectivity of the workflow manager and resource manager is performed.

The workflow manager also notifies the resource manager during its normal keep alive activities about node suspension, resumption, lock status, and cluster nodes which have been taken over. It also monitors the database connectivity of the resource manager.

One keep alive module must be configured with the name “`keep_alive`”.

The KeepAlive module can also be configured to setup a virtual IP address on unix platforms. The virtual IP address is configured when running ActivatorConfig. The if a cluster node crash the virtual IP address will be taken over by one of the other cluster nodes. The depending on the configuration of `auto_virtual_ip_takeover` the IP address

will automatically be taken over again by the original cluster node. This happens if set to true while if the parameter is set to false it is manual process which must be done via the user interface.

The KeepAlive module can also be configured to start a WatchDog process which will watch if Service Activator is running and if Service Activator crash the WatchDog process will restart Service Activator. The same will be the case if the WatchDog process crash then Service Activator will restart this process.

**Table 6-13 KeepAlive Module Parameters**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Reconfigurable</b>	<b>Default</b>
<i>keep_alive_time</i>	No	Configurable time at which the KeepAlive module must update the clusternodelist table to indicate that cluster node is active. The value is in milliseconds.	Yes	10000
<i>take_over_time</i>	No	Configurable time after which the KeepAlive module takes over a failed cluster node. Genrally, this would be twice the value of <i>keep_alive_time</i> .	Yes	30000
<i>job_startup_retry_count</i>	No	Configurable number of retry attempts that the KeepAlive module tries to startup a job after it decides to take over jobs from a failed cluster node.	Yes	3
<i>job_startup_retry_interval</i>	No	Configurable interval time in milliseconds between retry attempts to start a job.	Yes	10000
<i>update_heartbeat</i>	No	In standalone mode, this parameter can be set to false to indicate that normal processing of updating heartbeat time and monitoring of other cluster nodes should not be performed.	No	True

**Table 6-13 KeepAlive Module Parameters (Continued)**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Reconfigurable</b>	<b>Default</b>
<i>monitor_wait_interval</i>	No	Configurable interval time in milliseconds that the KeepAlive module will wait from the start of its initiation or after losing the database connectivity, before it will start monitoring of other cluster nodes.	No	30000
<i>db_poll_interval</i>	No	Configurable interval time in milliseconds between attempts to get a db connection when the KeepAlive module polls the database during db failure.	Yes	10000
<i>retrieve_jobs_buffer_size</i>	No	The number of jobs retrieved at a time during failover. To avoid memory issues only a small chunk of jobs for the failed cluster node are retrieved at a time.	Yes	The default value is set to half the size of Max-Work-List-Length.
<i>configure_virtual_ip</i>	Yes	Indicates if the Keepalive module during startup must try to set the configured virtual ip if any.	No	None
<i>auto_virtual_ip_taker</i>	No	Indicates if the cluster node during startup, must automatically take over its virtual ip if it is set up in another node in the cluster.	No	true
<i>virtual_ip_ping_timeout</i>	No	Specifies the maximum amount of time the cluster node will try to ping and wait for a reply, to check if its configured virtual ip address is set up in another node in the cluster. The default value is 10000 milliseconds.	No	10000 milliseconds



**Table 6-13 KeepAlive Module Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
start_watch_dog_process	No	Indicates if the watch dog process must be started. Only relevant on unix platforms. It is not support on Windows.	No	true
watch_dog_poll_interval	No	Configurable interval at which the watch dog process started by KeepAlive module will check if Service Activator is running	No	30000 milliseconds

**Example 6-12 KeepAlive Module**

```

<Module>
  <Name>keep_alive</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.KeepAliveModule</Class-Name>
    <Param name="Keep_alive_time" value="10000"/>
    <Param name="take_over_time" value="30000"/>
    <Param name="job_startup_retry_count" value="3"/>
    <Param name="job_startup_retry_interval" value="10000"/>
    <Param name="update_heartbeat" value="true"/>
    <Param name="monitor_wait_interval" value="30000"/>
    <Param name="db_poll_interval" value="10000"/>
  </Module>

```

## LDAPAuthModule

`com.hp.ov.activator.mwfm.engine.module.umm.LDAPAuthModule`

The LDAP auth module authenticates users by looking them up with username and password in an LDAP directory service and authorizes each user to have a set of roles that are associated with the user's entry in the directory.

The module can work with a number of LDAP servers, which can be active or backup. If there are multiple active servers, the load will be shared among them. Backup servers will only be contacted if it is not possible to contact any of the active servers.

The LDAP auth module is based on the following assumptions about entries in the directory tree:

- There is a root entry used to authenticate the module (HPSA) as a client of the directory service when it binds to the service.
- HPSA users are represented by entries which are children of a single entry (the user parent), typically an organizationUnit entry with ou=People. A special entry can be used for the system user.
- All user entries must have an attribute whose value is the Service Activator username. The name of this attribute is configurable.
- User entries also contain the Service Activator password for the user. The name of the password attribute is 'userPassword'.
- Similarly, Service Activator roles are represented as entries which are children of a single entry (the role parent), typically an organizationUnit entry with ou=Roles.
- All role entries must have an attribute whose value is the Service Activator role name. The name of this attribute is also configurable.
- Each role entry has a multi-valued attribute (its name is configurable) with a value for each user who has the role. The value equals the distinguished name of the user entry.

The roles defined in the LDAP server and used by Service Activator must also be created through the User Management Interface. Only users which have this configuration are able to login to Service Activator. During installation of Service Activator the roles "admin" and "internal" are created for User Management. The "admin" and "internal" roles must be configured in the LDAP server to make it possible to enable the LDAPAuthModule. However it is possible to use role mapping and hence create a different role for the System User in the underlying operating system.

---

### NOTE

You can configure this module to use normal TCP communication or Secure Socket Layer (SSL/TLS) communication. If SSL is configured and the server certificate is not signed by a trusted authority you must add the certificate into Java's list of trusted certificates. Also, note that only server certificates are supported

---

Name this module "authenticator" in the configuration.

### See Also

- "Roles, Privileges, and Authentication" in the *HP Service Activator - System Integrator's Overview*

- “Configuring Authentication or Authorization” on page 68 in *HP Service Activator—Developing Plug-Ins and Compound Tasks*
- “Setting Roles” on page 29

**Table 6-14 LDAPAuthModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>mwfm_remote_url</i>	Yes	Used to indicate from where to get the remote Workflow Manager service	No	None
ldap_hostname_1, ldap_hostname_2, ... ldap_hostname_N	Yes at least one	Hostname of IP address of LDAP server	No	None
ldap_port_1, ldap_port_2, ... ldap_port_N	Yes at least one		No	None
ldap_active_1, ldap_active_2; ... ldap_active_N	Yes at least one	"true" for active servers, "false" for backup servers. One server must be configured as active.	No	None
bindDN	Yes	Distinguished name of root entry, used to bind to the LDAP server.	No	None
bindCredential	Yes	Password for bindDN. Must be encrypted with the crypt utility which can be found in \$ACTIVATOR_OPT/bin.	No	None
userDN	Yes	Distinguished name of user parent entry.	No	None
userFilter	Yes	Name of the attribute which contains the user name to login to Service Activator	No	None

**Table 6-14 LDAPAuthModule Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
systemUser DN	No	Distinguished name of user parent entry. Only used when authenticating the system user. This can be used if another entry should be used for the system user compared with ordinary users	No	The value set for userDN
systemUser Filter	No	Name of the attribute which contains the user name to login to Service Activator. Only used when authenticating the system user. This can be used if another entry should be used for the system user compared with ordinary users	No	The value set for userFilter
rolesDN	Yes	Distinguished name of role parent entry.	No	None
roleFilter	Yes	Name of the multi-valued attribute on a role entry that identifies the users who have the role. Suggestion: member.	No	None
roleAttribute	Yes	Name of the attribute on a role entry that holds the name of the role. Suggestion: cn.	No	None
ssl	No	To use TLS/SSL encryption, set the value to "true".	No	false
showpassword	No	Display password in Log file. Must run with log level DEBUG.	No	false
searchTime Limit	No	Time limit to search and retrieve the username and password from the LDAP. Specifies the timeout value in milliseconds.	No	10000

**Table 6-14 LDAPAuthModule Parameters (Continued)**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Reconfigurable</b>	<b>Default</b>
checkForEmptyPassword	No	If set to true the auth module will check for if the provided password when doing user authentication is empty or not. If empty then access to HPSA will be denied	No	false

### Example 6-13 LDAPAuthModule Code

This example configures the LDAPAuthModule with the workflow Manager service.

```
<Module>
  <Name>authenticator</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.umm.LDAPAuthModule
  </Class-Name>
  <Param name="mwfm_remote_url" value="//localhost:2000/wfm"/>
  <Param name="ldap_host_1" value="localhost">
  <Param name="ldap_port_1" value="389"/>
  <Param name="ldap_active_1" value="false"/>
  <Param name="bindDN" value="cn=Manager,dc=my-domain,dc=com"/>
  <Param name="bindCredential" value="cr1DYotDW0l2NISQkSevtg=" />
  <Param name="userDN" value="ou=People,dc=my-domain,dc=com"/>
  <Param name="userFilter" value="uid"/>
  <Param name="rolesDN" value="ou=Roles,dc=my-domain,dc=com"/>
  <Param name="roleFilter" value="member"/>
  <Param name="roleAttribute" value="cn"/>
  <Param name="showpassword" value="false"/>
  <Param name="checkForEmptyPassword" value="false"/>
  <Param name="searchTimeLimit" value="2000"/>
</Module>
```

## LinuxAdvancedAuthModule

`com.hp.ov.activator.mwfm.engine.module.umm.LinuxAdvancedAuthModule`

The module provides authentication and authorization functionality based on the underlying Operating System authentication mechanism. It is only suitable for use on Linux.

The roles defined in the operating system and used by Service Activator must also be created through the User Management Interface. Only users which have this configuration are able to login to Service Activator. During installation of Service Activator the roles “admin” and “internal” are created for User Management. The “admin” role must be configured in the operating system to make it possible to enable the LinuxAdvancedAuthModule. However it is possible to use role mapping and hence create a different role for the System User in the underlying operating system.

Name this module “authenticator” in the configuration.

### See Also

- “Roles, Privileges, and Authentication” in the *HP Service Activator - System Integrator’s Overview*
- “Configuring Authentication or Authorization” on page 68 in *HP Service Activator—Developing Plug-Ins and Compound Tasks*
- “Setting Roles” on page 29

**Table 6-15 LinuxAdvancedAuthModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<code>mwfm_remote_url</code>	Yes	Used to indicate from where to get the remote Workflow Manager service	No	None
<code>sleep_time</code>	No	The time between the internal role cache is cleared.	No	30 min
<code>eight_char_password</code>	No	If password should be truncated to 8 characters before authentication is done	No	false
<code>secure_username</code>	No	Transform the user name to a valid value, cutting the user name from the first invalid character.	No	false

**Example 6-14 LinuxAdvancedAuthModule Code**

This example configures the LinuxAdvancedAuthModule with the workflow Manager service and the valid roles `activ_admin` or `activ_oper`

```
<Module>
  <Name>authenticator</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.umm.LinuxAdvancedAuthModule
  </Class-Name>
  <Param name="mwfm_remote_url" value="//localhost:2000/wfm"/>
  <Param name="validroles" value="activ_admin, activ_oper"/>
</Module>
```



## LoadFactorDistModule

`com.hp.ov.activator.mwfm.engine.module.LoadFactorDistModule`

The module allows the Workflow Manager to perform load balancing of workflow execution based on the load factor that is configured for each node in the cluster.

The configuration parameter “`load_factor`” determines the sequence in which the distribution of jobs is made among the cluster nodes. The `dispatch_local` parameter decides whether load balancing has to be performed or not. However the load balancing will be done in case the jobs should redistributed to one of the other cluster nodes in case of failover.

After a list of active nodes is retrieved, the jobs are distributed based on the configured load factor. Jobs are distributed to a cluster node till its load factor is reached, and subsequently they are distributed to the next node in the list. This distribution process is repeated until the last node in the cluster is reached, and then it begins from the first node in the list.

If a job fails to start, or the cluster node gets suspended or locked in-between the times when a request is dispatched and it reaches the cluster node, an attempt is made to start the job in the next cluster node that is in the list.

For example, if A, B, and C are three nodes in a cluster with load factor or 2, 3 and 2 respectively, the distribution sequence would be as follows:

A, A, B, B, B, C, C, A, A, B, B, B, C, C.....

**Table 6-16 LoadFactorDistModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<code>load_factor</code>	Yes	The configurable value that determines the sequence in which the distribution of jobs is made among the cluster nodes. Jobs are distributed to a cluster node till its load factor is reached, and subsequently they are distributed to the next node in the list.	Yes	Not applicable
<code>dispatch_local</code>	No	Configurable value that decides whether load balancing has to be performed or not. If this parameter is set to true, then the load balancing is switched off and jobs are dispatched only to the local cluster node. However the load balancing will be done in case the jobs should redistributed to one of the other cluster nodes in case of failover.	Yes	False

**Example 6-15**    **LoadFactor Distribution Module**

```
<Module>
  <Name>distribution_module</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.LoadFactorDistModule
  </Class-Name>
  <Param name="load_factor" value="10"/>
</Module>
```

## LogSearchModule

`com.hp.ov.activator.mwfm.engine.module.LogSearchModule`

The module creates log index files for which the Log Search UI component can be used.

The log modules must be configured to use the Log Search UI and in addition the index parameter must be set to true for the log modules which should perform indexing..

The module must be named `log_search_module`.

### See Also

- “XMLLogModule” on page 448 and “SolutionXMLLogModule” on page 434.

**Table 6-17** LogSearchModule Parameters

Parameter	Required	Description	Default
<i>commit_interval</i>	No	The maximum amount of milliseconds between two consecutive commit operations to the log file index (default is 30 seconds). To optimize performance, the <code>commit_interval</code> should not be set to too small values. Typically, the default value (or higher) will suffice. Please note that log statements cannot be searched before they have been committed to the index.	30 seconds
<i>commit_max_pending</i>	No	The maximum number of log messages to queue before committing them to the index (default is 500). Please note that log statements cannot be searched before they have been committed to the index.	500
<i>field_aliases</i>	No	A comma-separated list of mappings between English log index field names and localized field names. If this parameter is defined, localized field names can be used in advanced search operations.	None

**Table 6-17 LogSearchModule Parameters (Continued)**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>
<i>display_first_day_of_week</i>	No	This parameter must be set to an integer in the range 1 to 7. The parameter is used to define which day of the week to display first in the UI. The value are: 1 = Sunday, 2 = Monday, ..., 7 = Saturday	None
<i>use_24_hour_format</i>	No	Set this parameter to "true" (default) to use 24-hour format and to "false" to use 12-hour format on the UI.	"true"
<i>date_format</i>	No	This parameter is used to configure the date format to be used on the UI	YYY-MM-dd
<i>max_search_results</i>	No	A comma-separated list of integers that define the possible number of results to retrieve during a log search operation.	50, 100, 200, 500

## MailHook

`com.hp.ov.activator.mwfm.engine.module.MailHook`

The module sends e-mail messages to distribution lists when new messages are posted to certain queues.

There may be multiple hook modules configured within the Workflow Manager. Each hook module will be informed of the new arrival of a message.

This happens in an order determined from the module names. Each hook module must be given the name “hookN”, where N is a number indicating the order in which the modules are informed of new messages.

Thus, if there is only one hook module configured, it must be named “hook0”. A second hook module is then named “hook1,” and so on.

The module does not support re-configuration.

### See Also

- “Writing New Queue Hook” on page 476 for more information about Queue Hook modules.

Table 6-18

MailHook Parameters

Parameter	Required	Description	Default
<i>smtp_server</i>	Yes	The name of the SMTP server to be used for delivering the mail messages.	None
<i>mail_account</i>	Yes	The mail account that is used to send the messages. It should be something of the form login@domain.	None
<i>domain</i>	Yes	The domain that is used when talking to the SMTP server.	None
<i>queue0,</i> <i>queue1...</i> <i>queueN</i>	Yes	Specify <i>queue0</i> at a minimum. This parameter sets the name of the queue that will send an e-mail notification to the addresses contained in the <i>receiver0</i> text file.	None
<i>receiver0,</i> <i>receiver1...</i> <i>receiverN</i>	Yes	Names of the text files that contain the different e-mail addresses to send messages to when receiving messages in the queues specified by means of the queue parameters. The format for the file is one single e-mail address per line.	None

**Table 6-18**      **MailHook Parameters (Continued)**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>
<i>html</i>	No	If set to “false”, messages are sent as plain text. Otherwise by default they are sent as HTML.	“true”

## Monitor

`com.hp.ov.activator.mwfm.engine.module.monitor.Monitor`

This module collects statistics about functioning of the Workflow Manager. This data can be used to understand the engine performance. Statistics gathered include CPU time, wait time, number of running workflows, number of user logged in, number of worker threads, and number of activation threads.

Configure this module with the name “monitor”.

The module does not support re-configuration.

### See Also

- “Statistical Reports” on page 132 in *HP Service Activator- Introduction and Overview* for details on how you can use data collected.

**Table 6-19**      **Monitor Parameters**

Parameter	Required	Description	Default
<i>database_module</i>	Yes	The database module to use for writing audit records. If the database module is not specified, “db” is used.	“db”
<i>quantum</i>	Yes	Specifies the time in seconds between two consecutive measurements. A very short time interval may slow down your system performance. The minimal value is 60 seconds.	None
<i>store_statistics</i>	No	Indicates whether the Workflow Manager statistics is stored.	“false”

**Example 6-16**      **Monitor**

```
<Module>
  <Name>Monitor</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.monitor.Monitor
  </Class-Name>
  <Param name="quantum" value="10" />
</Module>
```





## NARequestModule

`com.hp.ov.activator.mwfm.engine.module.narequest.NARequestModule`

NARequestModule enables workflows to execute request on NA. The workflows must use the nodes specially developed for interacting with NA.

If you choose to configure multiple NA modules, give each module a unique name.

**NOTE**

The password used to establish the connection can be encrypted. Use the crypt utility to encrypt the password.

The module support NA version 9.1.

**Table 6-20 NARequestModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
na_username	Yes	The NA server user name for authentication.	No	None
na_password	Yes	The NA server password for authentication.	No	None
encrypted_password	No	Specifies if the provided na_password is encrypted.	No	false
na_ws_url	Yes	NA server URL for WS connection. Several URLs can be specified separated by ';' if NA is running in multi-master mode so they can be used for fall-back when opening a new session if connection fails. URLs are http URLs. This parameter should contain the same number of URL as "na_ejb_url".	No	None
na_ejb_url	Yes	NA server URL for RMI connection. Several URLs can be specified separated by ';' if NA is running in multi-master mode so they can be used for fall-back when opening a new session if connection fails. URLs are in the form <hostname>:<port>.	No	None

**Table 6-20 NARequestModule Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
queue_class	No	<p>This can be set to the <code>com.hp.ov.activator.mwfm.module.WeightedEngineQueue</code>, <code>com.hp.ov.activator.mwfm.module.SimpleEngineQueue</code>, or <code>com.hp.ov.activator.mwfm.module.PriorityEngineQueue</code></p> <p>The <code>WeightedEngineQueue</code> use the <code>PRIORITY</code> case-packet variable in a weighted way to prioritize the items on which the activation threads operate. Items that have the same priority will be processed in FIFO order.</p> <p>The <code>SimpleEngineQueue</code> will not do any prioritization of activation requests. They will be processed in FIFO order.</p> <p>The <code>PriorityEngineQueue</code> uses the <code>PRIORITY</code> case-packet variable to prioritize the items on which the activation threads operate. Items that have the same priority value will be processed in FIFO order.</p>	No	<code>com.hp.ov.activator.mwfm.engine.module.WeightedEngineQueue</code>
queue_name	No	The name of the queue the job will be waiting in while NA is processing a request	No	<code>external_request_queue</code>
retry_count	No	Number of times to retry processing of the external request.	No	3
retry_interval	No	Number of times to retry processing of the external request.	No	3
min_threads	No	Number of times to retry processing of the external request.	No	1
max_threads	No	Number of times to retry processing of the external request.	No	3



## NNMRequestModule

`com.hp.ov.activator.mwfm.engine.module.nnmrequest.NNMRequestModule`

NNMRequestModule enables workflows to execute request on NNM. The workflows must use the nodes specially developed for interacting with NNM.

You can configure the NNMRequestModule to access one NNM server.

If you choose to configure multiple NNM modules, give each module a unique name.

**NOTE**

The password used to establish the connection can be encrypted. Use the crypt utility to encrypt the password.

The module supports NNMi v9.10.

**Table 6-21 NNMRequestModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
nnm_username	Yes	The NNM server user name for authentication.	No	None
nnm_password	Yes	The NNM server password for authentication.	No	None
nnm_pass_is_encrypted	No	Specifies if the provided nnm_password is encrypted.	No	false
nnm_hostname	Yes	The host name of the NNM server.	No	None
nnm_protocol	No	The protocol to be used for the connection to the NNM server (either HTTP or HTTPS).	No	HTTP
nnm_port	No	The NNM server port.	No	80 for HTTP and 443 for HTTPS
nnm_keystore	No	Path to the keystore containing the NNMi certificate (used on HTTPS connections).	No	\$ACTIVATOR_ETC/config/mwfmSSL.keystore
nnm_keystore_password	No	The password for accessing and recovering keys from the KeyStore.	No	changeit

**Table 6-21 NNMRequestModule Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
queue_class	No	<p>This can be set to the <code>com.hp.ov.activator.mwfm.module.WeightedEngineQueue</code>, <code>com.hp.ov.activator.mwfm.module.SimpleEngineQueue</code>, or <code>com.hp.ov.activator.mwfm.module.PriorityEngineQueue</code></p> <p>The <code>WeightedEngineQueue</code> use the <code>PRIORITY</code> case-packet variable in a weighted way to prioritize the items on which the activation threads operate. Items that have the same priority will be processed in FIFO order.</p> <p>The <code>SimpleEngineQueue</code> will not do any prioritization of activation requests. They will be processed in FIFO order.</p> <p>The <code>PriorityEngineQueue</code> uses the <code>PRIORITY</code> case-packet variable to prioritize the items on which the activation threads operate. Items that have the same priority value will be processed in FIFO order.</p>	No	<code>com.hp.ov.activator.mwfm.engine.module.WeightedEngineQueue</code>
queue_name	No		No	<code>external_request_queue</code>
retry_count	No	Number of times to retry processing of the external request.	No	3
retry_interval	No	Number of times to retry processing of the external request.	No	3
min_threads	No	Number of times to retry processing of the external request.	No	1
max_threads	No	Number of times to retry processing of the external request.	No	3



## OVOMessageModule

`com.hp.ov.activator.mwfm.engine.module.OVOMessageModule`

The module allows the Workflow Manager to send messages to OVO. It assumes that the OVO agent software has been installed on the local machine and uses the `opcmsg` command to send the message to the OVO server.

You can send messages from a workflow to OVO by using the `SendAlarm` node. DO NOT confuse this node with the `SendMessage` node, which is designed to send messages via a different kind of module, such as the `SocketSenderModule`. For more information about sending messages to OVO, see the description of “SendAlarm” on page 289.

A number of the message settings, such as severity, have the default values that can be overridden when a message is sent. See the description of “SendAlarm” on page 289 for details about setting these parameters when a message is sent.

This module does not support re-configuration.

**Table 6-22** OVOMessageModule Parameters

Parameter	Required	Description	Default
<code>opcmsgcommand</code>	Yes	Used to specify the path to the <code>opcmsg</code> command.	None
<code>application</code>	No	Used to specify a default application name for messages sent by this module. The caller of the <code>sendMessage()</code> method can override the caller of the application.	“Service Activator”
<code>msg_grp</code>	No	Used to specify a default message group for messages sent by this module. The caller of the <code>sendMessage()</code> method can override the <code>msg_grp</code> .	“Misc”

**Example 6-17** OVOMessageModule

```
<Module>
  <Name>ovo_sender</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.OVOMessageModule
  </Class-Name>
  <Param name="opcmsgcommand"
    value="/usr/OV/bin/OpC/opcmsg.exe"/>
  <Param name="application" value="ServiceActivator"/>
  <Param name="msg_grp" value="misc"/>
</Module>
```

## QueueDistModule

`com.hp.ov.activator.mwfm.engine.module.QueueDistModule`

The module allows the Workflow Manager to perform load balancing of workflow execution based on the number of jobs currently running in the cluster nodes.

The `dispatch_local` parameter decides whether load balancing has to be performed or not. However the load balancing will be done in case the jobs should be redistributed to one of the other cluster nodes in case of failover.

After a list of active nodes is retrieved, the cluster node with least number of currently running jobs is selected and workflow execution is assigned to it.

If a job fails to start, or the cluster node gets suspended or locked in-between the times when a request is dispatched and it reaches the cluster node, an attempt is made to start the job in the next cluster node that is in the list.

**Table 6-23** QueueDistModule Parameters

Parameter	Required	Description	Default
<code>dispatch_local</code>	No	Configurable value that decides whether load balancing has to be performed or not. If this parameter is set to true, then the load balancing is switched off and jobs are dispatched only to the local cluster node. However the load balancing will be done in case the jobs should be redistributed to one of the other cluster nodes in case of failover.	Yes

**Example 6-18** Queue Distribution Module

```
<Module>
  <Name>distribution_module</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.QueueDistModule
  </Class-Name>
  <Param name="dispatch_local" value="false"/>
</Module>
```





## QueueDBPersistenceModule

```
com.hp.ov.activator.mwfm.engine.module.queueing.QueueDBPersistenceModule
```

This module must be configured when the queueing sub-system must be used.

Jobs can be queued by using the workflow manage rmi method startQueueJob. The module is used by the module named queue\_module.

The following is the configuration for the queue db persistence module. The name of the module must "queue\_persistence\_module".

### See Also

- “QueueModule” on page 419

### Example 6-19

#### QueueModule

```
<Module>  
  <Name>queue_module</Name>  
  <Class-Name>  
    com.hp.ov.activator.mwfm.engine.module.queueing.QueueDBPersistenceModule  
  </Class-Name>  
</Module>
```

---

## QueueModule

`com.hp.ov.activator.mwfm.engine.module.queuing.QueueModule`

This module must be configured when the queuing sub-system must be used.

Jobs can be queued by using the workflow manage rmi method `startQueueJob`. The module require the module `queue_persistence_module` to be configured.

The following is the configuration for the queue module. The name of the module must "queue\_module".

### See Also

- “QueueDBPersistenceModule” on page 418

### Example 6-20

#### QueueModule

```
<Module>
  <Name>queue_module</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.queuing.QueueModule
  </Class-Name>
</Module>
```



## RoundRobinDistModule

`com.hp.ov.activator.mwfm.engine.module.RoundRobinDistModule`

The module allows the Workflow Manager to perform load balancing of workflow execution requests in a round robin fashion.

The configuration parameter *dispatch\_local* decides whether load balancing has to be performed or not. However the load balancing will be done in case the jobs should redistributed to one of the other cluster nodes in case of failover.

After a list of active nodes is retrieved, the jobs are distributed in a sequential fashion, such that the workflow execution is equally distributed among the cluster nodes.

If a job fails to start, or the cluster node gets suspended or locked in-between the times when a request is dispatched and it reaches the cluster node, an attempt is made to start the job in the next cluster node that is in the list.

For example, if A, B, and C are three nodes in a cluster, the distribution sequence is as follows:

A, B, C, A, B, C.....

### Example 6-21 RoundRobin Distribution Module

```
<Module>
  <Name>distribution_module</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.RoundRobinDistModule
  </Class-Name>
  <Param name="dispatch_local" value="false" />
</Module>
```

**Table 6-24 RoundRobinDistModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>dispatch_local</i>	No	Configurable value that decides whether load balancing has to be performed or not. If this parameter is set to true, then the load balancing is switched off and jobs are dispatched only to the local cluster node. However the load balancing will be done in case the jobs should redistributed to one of the other cluster nodes in case of failover.	Yes	False

## SchedulerModule

`com.hp.ov.activator.mwfm.engine.module.SchedulerModule`

This module allows you to start a workflow at a specified time in the future. The module contains a thread pool to handle the actual job execution at the specified time. If the scheduled job is configured to reoccur the module will start a new job with the given frequency. All jobs started will have unique Job Ids.

The scheduled jobs can be managed either via the RMI interface, via the special nodes or via the UI.

This module follows the Master-Slave approach. If the module is a master, it will have the privilege to schedule a job, and start the job at the scheduled time. However, if the module is a slave, it cannot schedule a job or start a scheduled job.

### See Also

- “ScheduleJob” on page 286
- “ModifyScheduledJob” on page 213
- “QueryScheduledJob” on page 256
- “DeleteScheduledJob” on page 136

**Table 6-25 SchedulerModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>max_threads</i>	Yes	The maximum amount of threads in the pool used for launching the jobs at the specified time.	Yes	None
<i>min_threads</i>	No	The minimum amount of threads in the pool of scheduled activation threads. These threads remain alive for as long as the module is used. If additional threads are required, the SchedulerModule creates dynamic threads up to the <i>max_threads</i> limit. Dynamic threads expire after 10 seconds of inactivity, or after the time specified in <i>idle_thread_keep_alive</i> .	Yes	<i>max_threads</i>
<i>idle_thread_keep_alive</i>	No	The amount of time (in seconds) for which an idle dynamic thread exists before it expires and is deleted (see the <i>min_threads</i> ). This only applies to dynamic threads.	Yes	10

**Example 6-22 SchedulerModule**

```
<Module>
  <Name>scheduler_module</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.SchedulerModule
  </Class-Name>
  <Param name="min_threads" value="1"/>
  <Param name="max_threads" value="2"/>
  <Param name="idle_thread_keep_alive" value="20"/>
</Module>
```

## SelfMonitoringModule

`com.hp.ov.activator.mwfm.engine.module.SelfMonitoringModule`

The self monitoring module monitors the health of the Service Activator system and raises alarm when a set threshold is violated. The parameters that are monitored by the self monitoring module are:

- Heap size - The module checks if the current heap size exceeded the threshold set. The threshold heap size is set as a percentage value indicating the percentage of maximum heap size which should not be exceeded. OID="1.3.6.1.4.1.11.2.52.1.1"
- Maximum worklist length - The module checks if the current work list length has exceeded the percentage of maximum work list length as configured in the threshold setting. OID = "1.3.6.1.4.1.11.2.52.2.1".
- Internal Suspension - The module generates traps/alarms when a node undergoes internal suspension. OID="1.3.6.1.4.1.11.2.52.5.1".
- Node down - When HPSA is running in a cluster. If one of the cluster nodes go down, a notification is generated. OID="1.3.6.1.4.1.11.2.52.5.2.1".

The SelfMonitoringModule can generate an SNMP trap, log alarms to a file and insert audit records to the database. These can be enabled individually in the module configuration.

The Self Monitoring Module will also save time series in a Round-Robin database. This information can be seen in the User Interface. Data regarding memory, worker threads, activation threads, activation queue size, total number of jobs, and finally user sessions will be saved.

**Table 6-26 SelfMonitoringModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>poll_intrval</i>	No	The polling interval at which the parameters are verified. This is defined in microseconds.	No	10000
<i>threshold_percent_heap_size</i>	No	This parameter indicates the percentage of heap size which will trigger an alarm. For example if it is defined as 80, an alarm/trap will be sent when the heap usage is 80% of the max heap size defined.	No	80
<i>threshold_percent_maxworklistlength</i>	Yes	This parameter indicates the percentage of max worklist length which will trigger an alarm. For example if it is defined as 80, an alarm/trap will be sent when the work list length reaches 80% of the max work list length defined.	No	80



**Table 6-26 SelfMonitoringModule Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
snmp_module	Yes, if send_snmp_trap is set to true	The snmp module to be used to send SNMP traps.	No	None
send_snmp_trap	No	This is a Boolean parameter which indicates whether SNMP traps should be sent. A value of true indicates that the traps should be sent and false otherwise	No	false
log_alarm	No	This is a boolean parameter indicating if a log entry should be written. A value of true indicates that a log entry is written.	No	false
max_alarm_entries	No	The maximum number of alarm entries that will be logged into a single file.	No	1000
audit_events	No	This is a Boolean parameter which indicates whether threshold violations should be inserted as audit records. A value of true indicates that the audit records should be inserted and false otherwise. If this parameter has not been defined then the default will be false. In addition to this auditing will depend on whether system wide auditing has been enabled or not.	No	false
granularity	No	A comma separated list which indicat which time series should be save in the RRD database maained and handled by the module	No	1,5,30,240,1440,10080
samples	No	The number of samples which are saved for each time serie.	No	360

**Example 6-23 SelfMonitoringModule**

```

<Module>
  <Name>db</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.SimpleDatabaseModule
  </Class-Name>
  <Param name="poll_interval" value="10000"/>
  <Param name="threshold_percent_heap_size" value="80"/>
  <Param name="threshold_percent_maxworklistlength" value="80"/>
  <Param name="snmp_module" value="snmp_sender"/>
  <Param name="send_snmp_trap" value="true"/>
  <Param name="log_alarm" value="true"/>
  <Param name="max_alarm_entries" value="1000"/>
  <Param name="audit_events" value="false"/>
</Module>

```

## SNMPSENDERMODULE

`com.hp.ov.activator.mwfm.engine.module.SNMPSENDERMODULE`

The module enable the Workflow Manager to to send snmp traps to an snmp manager.

The module supports sending both SNMP v2c and SNMP v3 traps.

The module can be accessed by the `SendSNMPTrap` workflow node and must also be configured if the `SelfMonitoringModule` is configured.

Multiple SNMP modules can be configured to send traps to multiple SNMP managers.

**Table 6-27** SNMPSENDERMODULE Parameters

Parameter	Required	Description	Default
<i>host</i>	Yes	host name of the SNMP manager.	None
<i>port</i>	No	Port in which the SNMP manager is listening on	162
<i>engine_id</i>	Yes, if SNMP version is 3	Engine Id of the agent.	None
<i>username</i>	Yes, if SNMP version is 3	User name to be used. Used for version 3 traps.	None
<i>password</i>	Yes, if SNMP version is 3	Password to be used. Used for version 3 traps. The password has to encrypted using the crypt utility.	None
<i>auth_protocol</i>	No	Authorization protocol to be used. It can either be MD5/SHA1. Used for version 3 traps.	MD5
<i>community</i>	No	Community to be used while sending the SNMP trap. Used for version 2 traps.	public
<i>privacy_protocol</i>	No	Protocol to be used for privacy. The value can either be DES/AES. Used for version 3 traps.	DES

**Table 6-27 SNMPSenderModule Parameters (Continued)**

Parameter	Required	Description	Default
<i>privacy_pwd</i>	No	Privacy password to be used.	In case the password parameter has been defined then the default value of <i>privacy_pwd</i> will be the same as that of the "password" parameter.
<i>snmp_version</i>	No	The snmp version to be used. Valid values are 2 and 3.	3

**Example 6-24 SNMP Sender Module**

```

<Module>
  <Name>SNMPSenderModule</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.SNMPSenderModule
  </Class-Name>
  <Param name="host" value="localhost"/>
  <Param name="port" value="162"/>
  <Param name="engine_id" value="800007e580fd791162bfae0042"/>
  <Param name="username" value="AuthUser"/>
  <Param name="password" value="caLbA9g6h1n1/lQXeTL/mQ==" />
  <Param name="community" value="public"/>
  <Param name="auth_protocol" value="MD5"/>
  <Param name="privacy_protocol" value="DES"/>
  <Param name="privacy_pwd" value="caLbA9g6h1n1/lQXeTL/mQ==" />
  <Param name="snmp_version" value="3"/>
</Module>

```

## SocketListenerModule

`com.hp.ov.activator.mwfm.engine.module.SocketListenerModule`

The module opens a socket on a specified port and waits for incoming messages. When a message arrives, the listener, either saves the message to a file or in the database based on the configuration and starts a new workflow to process the message.

The listener passes the location of the message received in case-packet variable called `message_url`, , the value being `data:<message>`, `file:<file path>` or `db:<message id>` respectively. A workflow that is to be started by the `SocketListenerModule` must have a variable by this name.

Configure as many `SocketListenerModules` as necessary. Configure each listener on a unique port and with a unique name. A listener can start any workflow as long as the workflow contains the `message_url` variable.

If the module is configured to save messages in a file, to avoid the risk of messages being accidentally overwritten or deleted, it is important to ensure that each `SocketListenerModule` configured in the `mwfm.xml` has its own directory to save arriving messages in.

If the messages are saved in a file, workflows are responsible for deleting such message files, typically using the `RemoveData` node. If the messages are written to a database, they must also be deleted after the completion of corresponding workflow execution using the `RemoveData` node. If the message is passed to the workflow with the data url (`data:message`) no deletion is needed.

The module is multi-threaded and avoids the blocking behavior. Incoming client socket connections are placed in a queue for processing by socket listener threads that are managed by this module.

Use the `recover` parameter when you configure the `SocketListenerModule` so that, when the Workflow Manager starts, the listener module looks for all existing message files or unprocessed messages stored in the database and start a new workflow to handle each outstanding file or message in the database. This is not necessary if the Workflow Manager is configured to maintain persistent state of workflows (this is the default configuration).

If the module is configured to save messages in a database, it looks for all unprocessed messages received by the failed cluster node and starts a new workflow to handle each outstanding message in failover scenarios.

By default, the `SocketListenerModule` adds an XML header at the beginning of a message written to a file. You can configure whether or not the module writes the header, which DTD the header refers to, and the root tag. The default header looks similar to this:

```
<?xml version="1.0" encoding="utf-8" />
<!DOCTYPE msg SYSTEM
"file://etc/opt/OV/ServiceActivator/config/exchange.dtd">
```

---

### NOTE

Messages received by the `SocketListenerModule` must be in UTF-8 format so that the Workflow Manager can process them properly.

---

**NOTE** You can configure this module to use normal TCP communication or Secure Socket Layer (SSL) communication. If you choose SSL, you must provide a valid SSL certificate identifying the server.

For additional information, see Appendix A, “Configuring Service Activator to Use Secure Socket Layer (SSL) Protocol,” on page 487.

---

**NOTE** If you use prioritized workflows, and you have a controller workflow configured for a `SocketListenerModule`, you must make sure that the priority of the controller workflow is at least as high as the priority of the workflows it starts. Otherwise, the controller workflow can be starved.

---

The module does not support re-configuration.

**See Also**

- “RemoveData” on page 277 for information about removing a file after a workflow has finished processing it.
- “SendMessage” on page 291 and “SocketSenderModule” on page 432 for information about sending messages back to a waiting program.
- “WorkManagerModule” on page 447 for information about workflow prioritization.

**Table 6-28** **SocketListenerModule Parameters**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>
<i>workflow</i>	Yes	The workflow to be started upon receipt of a message. A parameter with the name <code>message_url</code> (containing the file where the message is stored or the message id referring to a row in the table where the message is stored ) is passed through to the new workflow instance just created.	None
<i>port</i>	Yes	The port on which the module listens.	None

**Table 6-28 SocketListenerModule Parameters (Continued)**

Parameter	Required	Description	Default
<i>recover</i>	No	If set to "true", on start-up, all existing messages in the directory <code>received_messages</code> or unprocessed messages in the <code>DATABASE MESSAGE</code> table starts workflows. This is only useful if the Workflow Manager is not configured to maintain the state of running workflows. In that case, a workflow might have been started to process an incoming message, but the Workflow Manager shut down before it completed its processing of the message.	"false"
<i>directory</i>	No	The directory ( <code>\$ACTIVATOR_VAR/</code> ) where messages are held until processing.	<code>\$ACTIVATOR_VAR/received_messages</code>
<i>save_messages</i>	No	Enables message logging to the <code>\$ACTIVATOR_VAR/received_messages_log</code> directory.	"false"
<i>header</i>	No	Specifies whether the XML header should be put before the message being received or not. Possible values are <code>true</code> and <code>false</code> .	"true"
<i>dtd_root_tag</i>	No	If you specify this parameter, the header of the XML document is set to point to it. If it is not specified and a DTD is present, its value is <code>msg</code> .	<code>msg</code>
<i>dtd</i>	No	Specifies the document type definition that should be used for validating the XML message. You can specify the absolute location of the DTD file, or a path relative to <code>\$ACTIVATOR_ETC/config</code> . If you do not want to specify a DTD, set the value to "NO_DTD".	<code>exchange.dtd</code>

**Table 6-28 SocketListenerModule Parameters (Continued)**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>
<i>keystore</i>	No	The path of the SSL keystore to be used by the socket listener for identifying itself to clients. If you specify this parameter, an SSL connection is utilized.	None
<i>keystore_password</i>	No	The password to use to open the keystore.	None
<i>clientauth</i>	No	If you specify this parameter by setting it to "true", clients to this socket must present an SSL certificate to the socket listener. This is only available if the socket listener has been configured for SSL connections.	"false"
<i>min_threads</i>	No	The minimum number of threads to maintain for processing arriving messages.	1
<i>max_threads</i>	No	The maximum number of threads to maintain for processing arriving messages.	3
<i>write_message_to</i>	Yes	Indicates the storage medium of arriving messages, data, file, or database.	None

---

## SocketSenderModule

`com.hp.ov.activator.mwfm.engine.module.SocketSenderModule`

The module is used to configure a mechanism for sending messages through a TCP socket to a port on a remote (or local) server. Configure as many `SocketSenderModules` as necessary. Give each module a unique name.

When the Workflow Manager starts up, it initializes each configured `SocketSenderModule` to contact the `host/port` that has been configured.

---

### NOTE

You can configure this module to use normal TCP communication or Secure Socket Layer (SSL) communication. If you choose SSL, you must provide a valid SSL certificate identifying the server.

For additional information, see Appendix A, “Configuring Service Activator to Use Secure Socket Layer (SSL) Protocol,” on page 487.

The `read_message_from_db` parameter indicates whether the messages have to be read from the database or the file system.

---

### NOTE

Before using this module with the `fault_tolerant` parameter set to “true”, be sure that you have a program waiting at the given `host` and `port`.

With this parameter set to “true” the `SocketSenderModule` saves every message it attempts to send, if the parameter “`read_message_from_db`” is set to false, until the message can be sent successfully. If the “`read_message_from_db`” is true will the message in the database first be removed when the message is sent successfully.

If there is no program to receive the message, then the unsent messages occupy much disk space or database space while the module wastes CPU resources attempting to resend the messages.

If the parameter “`read_message_from_db`” is set to true, in a distributed setup, the pending unsent messages are also taken over by the cluster node that takes over the failed cluster node. The parameter “`read_message_from_db`” is also used during startup to read unsent messages either from the file system or the database.

The module does not support re-configuration.

### See Also

- “SendMessage” on page 291 for more information about sending messages from a workflow to a waiting program.

---

**Table 6-29**

**SocketSenderModule Parameters**

Parameter	Required	Description	Default
<i>host</i>	Yes	The host to send the messages to.	None



**Table 6-29 SocketSenderModule Parameters (Continued)**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Default</b>
<i>port</i>	Yes	The port in the host to send the messages to.	None
<i>fault_tolerant</i>	No	If “true”, messages that cannot be sent for any reason are saved and retried after a few seconds (see <i>sleep_time</i> ).	“false”
<i>sleep_time</i>	No	Use this parameter to specify a retry interval when <i>fault-tolerant</i> is set to true. Specify the number of seconds to wait before retry.	30
<i>keystore</i>	No	Use this parameter to send messages using SSL communication. This parameter specifies the path to the SSL certificate to be used. If the parameter is not specified, normal TCP protocol is used to send messages.	None
<i>keystore_password</i>	No	The password to use to open the keystore.	None
<i>pending_message_directory</i>	No	Sets the pending messages directory. If you specify this parameter, you will override the default directory <i>\$ACTIVATOR_VAR/pending_messages</i> . This is useful if you have multiple SocketSenderModules configured.	None
<i>read_data_from_db</i>	No	Indicates the storage medium for reading messages, file or database. If the value is true the messages will be read from the database.	true

## SolutionXMLLogModule

`com.hp.ov.activator.mwfm.engine.module.XMLLogModule`

The Workflow Manager requires a module called `log_manager` to provide logging functionality. `XMLLogModule` is the preferred class for this purpose. However, for having solution-specific log files apart from the existing MWFM log files, `SolutionXMLLogModule` can be configured by specifying different module names.

The module logs messages to files in the directory `$ACTIVATOR_VAR/<hostname>/log`, the file name specified by the `solution_name` parameter. The module also performs automatic rollover of log files after it has logged the configured number of messages.

### See Also

- “Log” on page 204

**Table 6-30** SolutionXMLLogModule Parameters

Parameter	Required	Description	Reconfigurable	Default
<code>log_max_entries</code>	No	The maximum number of log entries written to a log file before the log file is closed and a new one is created.	Yes	1000
<code>log_level</code>	No	The following <code>log_level</code> parameters set the type of information logged: ERROR - to record only error messages logged . WARNING - to record errors and warnings. INFORMATIVE - to record errors, warnings and some additional information. DEBUG - to get additional debugging information . DEBUG2 - to get even more detailed debugging information.	Yes	INFORMATIVE
<code>log_allow_statistics</code>	No	A value of “true” allows the logging of statistical data. This is distinct from the level of logging that is chosen.	No	“false”
<code>log_directory</code>	Yes	The location where the log files will be stored under the hostname folder.	No	<code>VAR_HOME/log</code>
<code>solution_name</code>	Yes	Specifies the name of the solution that is used to generate log files.	No	None

**Table 6-30 SolutionXMLLogModule Parameters (Continued)**

Parameter	Required	Description	Reconfigurable	Default
<i>log_max_files</i>	No	The maximum number of log files that can exist. If the max number is reached the first file will be overwritten. The log module will always starts with creating log file with the number 0. So the log module will rotate the log files over time. By setting this option you can avoid that the size of the log files grows indefinitely. If the value is set to 0 (default value) no log rotation is done.	No	0
<i>index</i>	No	This optional parameter determines whether log messages will be indexed for searching or not. A value of "true" will enable indexing, while "false" (default value) or any other value will disable it	Yes	false

**Example 6-25 SolutionXMLLogModule Code**

```

<Module>
  <Name>test_solution_manager</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.SolutionXMLLogModule
  </Class-Name>
  <Param name="log_level" value="INFORMATIVE"/>
  <Param name="log_max_entries" value="1000"/>
  <Param name="log_allow_statistics" value="true"/>
  <Param name="log_directory" value="C:/HP/OpenView/ServiceActivator/var/log"/>
  <Param name="solution_name" value="test_solution"/>
</Module>

```

## SyncModule

`com.hp.ov.activator.mwfm.engine.module.SyncModule`

The module handles parent-child workflow synchronization. All the parent and child workflows communicate via the Sync Module and there will be no direct interaction between them. The Sync module makes it possible for the parent workflow to wait for one or more children's response at the same time. Parent workflow communicates with the sync module through the AskFor node and the child workflow responds to the parent workflow either through Sync node or through the SyncHandler end handler. The synchronization will happen through a queue that is configured in the AskFor node and the Sync node / SyncHandler end handler.

The Sync Module allows the workflow manager to perform the following.

- Determine when to send the synchronization responses to the waiting parent workflow. This depends on the waiting condition configuration of the AskFor node. Please refer to the AskFor node documentation to understand more on this parameter.
- Keep the children waiting if the parent workflow has not entered into the synchronization process. The child workflow will be parked in the queue that is configured in the Sync node.
- Retry the child response whenever there is a synchronization failure. This is primarily in the distributed environment. The synchronization could fail due to the following reasons.
  - Loss of Database Connectivity
  - Network failure
  - Node being suspended (operator)
  - Node shutting down
- Retry wake up of children whenever there is a wake up failure. Wakeup failure could be due to the following reasons
  - Network failure
  - Loss of Database Connectivity
  - Node where the child is waiting is suspended
  - Node where the child is waiting is shutting down
  - Unable to persist the data using File Transaction module

Sync Module gives the facility to make the parent workflow wait on

- All the children workflows that it spawns
- A combination of the children workflows that it spawns
- A "number" of children workflows that it spawns - count of workflows can be specified instead of the actual workflow job IDs.
- Any one of the children workflows that it spawns

The Sync Module will cache the response from the children in the database depending on the parent workflow's waiting condition that is configured as part of the AskFor node. Please refer to AskFor node documentation to know more about the parent waiting condition.

The scenarios where the child response will be cached in the database and the child workflow parked in the configured queue (queue configured in the Sync node) are:

- "Parent workflow waiting in the AskFor node and the responding child is not the last child (waiting condition of ALL / COUNT)
- "Parent workflow entering the AskFor node for synchronization after the children have responded (waiting condition of ALL / COUNT / ANY)

The scenarios where the child response will not be cached in the database (and will not be parked anywhere) are:

- "Parent workflow waiting in the AskFor node for synchronization and the responding child is the last child (waiting condition of ALL / COUNT) or the first child (waiting condition of ANY).
- "Parent waiting in the AskFor node for synchronization and the responding child is the only child (waiting condition where only one workflow is configured in ALL case or count of workflows is set to 1 in case of waiting condition of COUNT)

The child workflows that are sending their synchronization responses through the SyncHandler end handler will never be parked in any queue even if the parent workflow's waiting condition is not met or the parent workflow is not found waiting for a response.

When the synchronization is complete, the data in the database will be cleaned up in addition to waking up of the waiting children.

There are different exception situations where the sync module's monitor threads will come into play. There are three monitor threads

- WakeUpMonitor - This thread will be used whenever the sync module couldn't wake up the parent or the child workflow. The wake up activity will be parked in this thread and this thread will retry till the wake up is successful.
- DBCleanUpMonitor - This thread will be used whenever the sync module couldn't clean up records in the child and child\_response table.
- ParentSyncFailureMonitor - This thread will be used whenever the child workflow (through sync module) couldn't send a response to the parent workflow.

The scenarios where the workflows will be failed are listed below:

- When more than one child workflow is responding to the parent workflow using the same case packet variable, the parent workflow along with the waiting child workflows will be failed as this is a workflow design issue.
- When there are children workflows waiting for the parent workflow to synchronize and the parent workflow never attempted synchronization, those waiting children workflows will be failed at the end of the parent workflow.
- When a child workflow is attempting synchronization and the corresponding parent workflow is not running or the queue that is used to synchronize is not found, the child workflow will be failed

- If the sync module encounters SQLException during the parent or child registration, the parent workflow along with the associated (only those registered in the sync module) will be failed.
- Parent workflow and any children waiting in the sync module will be failed if the parent workflow's AskFor node is not configured properly. Please refer to the AskFor node page for more information.

**See Also**

- “Roles, Privileges, and Authentication” in *HP Service Activator System Integrator’s Overview*
- *HP Service Activator Developing Plug-Ins and Compound Tasks*
- “Setting Roles” on page 29
- “AskFor” on page 104

**Table 6-31 SyncModule Parameters**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Reconfigurable</b>	<b>Default</b>
wakeup_monitor_interval	No	Configurable time interval that the Wake Up Monitor thread will sleep after every check to see if there are any pending wake up work. The value is in milliseconds.	Yes	4000
db_cleanup_interval	No	Configurable time interval that the DB Cleanup Monitor thread will sleep after every check to see if there are any pending clean up. The value is in milliseconds.	Yes	4000
parent_notification_interval	No	Configurable time interval that the Parent Sync Failure Monitor thread will sleep after every check to see if there are any pending parent synchronization. The value is in milliseconds	Yes	2000

**Example 6-26**      **SyncModule Code**

This example configures the SyncModule.

```
<Module>
  <Name>authenticator</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.SyncModule
  </Class-Name>
  <Param name="wakeup_monitor_interval" value="4000"/>
  <Param name="db_cleanup_interval" value="4000"/>
  <Param name="parent_notification_interval" value="2000"/>
</Module>
```





## UCMDBRequest Module

`com.hp.ov.activator.mwfm.engine.module.UCMDBRequestModule`

UCMDBRequestModule enables workflows to communicate with an uCMDB server. This module will enable communication with the UCMDB through the web services API exposed by it.

This module serves as the web service client to the web. The following web services of UCMDB can be executed:

- createCIsAndRelations
- updateCIsAndRelations
- deleteCIsAndReleations
- getCIsByid
- getCIsByType
- getCINeighbours
- executeTopologyQueryByName
- executeTopologyQueryByNameWithParameters
- executeTopologyQueryWithParameters

Workflows must use the uCMDB nodes to get access to the web methods.

### See Also

- For the node descriptions, see Chapter 4 of this guide.

This module is inherently multithreaded and the minimum and maximum threads can be configured as part of the module configuration. This module is invoked in an asynchronous fashion. After the node invokes this module, the module puts the node in a queue and notifies the node after the response is received from uCMDB

A single UCMDBRequestModule is configured as follows.

The module supports uCMDB 8.0.

**Table 6-32 UCMDBRequestModule Parameters**

Parameter	Required	Description	Default
<i>wsdl_url</i>	Yes	The URL of the deployed UCMDB wsdl	<i>none</i>
<i>username</i>	No	Username to connect to the uCMDB	None

**Table 6-32 UCMDBRequestModule Parameters (Continued)**

Parameter	Required	Description	Default
<i>password</i>	No	Password to connect to the uCMDB. Should be specified in encrypted format. Encryption of the password must be done using the crypt command in \$ACTIVATOR_BIN	<i>None</i>
<i>min_threads</i>	No	The minimum number of threads that will be created to process uCMDB requests	1
<i>max_threads</i>	No	The maximum number of threads that will be created to process uCMDB requests. This is the number of simultaneous requests that can be processed. Other incoming requests will be queued until one of the threads becomes available.	3
<i>queue_name</i>	No	Name of the queue in which the waiting jobs will be put.	<i>ucmdb</i>
<i>queue_class</i>	No	This is the name of the java class that implements the queue for the work items. Valid values are com.hp.ov.activator.mwfm.engine.module.SimpleEngineQueue and com.hp.ov.activator.mwfm.engine.module.PriorityEngineQueue	com.hp.ov.activator.mwfm.engine.module.SimpleEngineQueue

**Example 6-27 UCMDBRequestModule**

```

<Module>
  <Name>ucmdb_request</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.UCMDBRequestModule
  </Class-Name>
  <Param name="wsdl_url"
value="http://localhost:5050/axis2/services/UcmdbService?wsdl"/>
  <Param name="user_name" value="admin"/>
  <Param name="password" value="ux8/AgHpqL8c7v6Nw668VA==" />
  <Param name="min_threads" value="1"/>
  <Param name="max_threads" value="3"/>
  <Param name="queue_class"
value="com.hp.ov.activator.mwfm.engine.module.SimpleEngineQueue"/>
</Module>

```

## UsageMonitoringModule

`com.hp.ov.activator.mwfm.engine.module.UsageMonitoringModule`

This module collects usage information (service requests) and stores it in the form of usage records in the system database. The UsageMonitoringModule cannot be disabled.

The name of the module must be "usage\_monitoring\_module".

The module does not support re-configuration.

**Table 6-33 UsageMonitoringModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<i>collection_interval</i>	No	The interval (in seconds) between two consecutive usage data collection events. The minimum allowed value for this parameter is 1 minute (60 seconds).	No	3600
<i>usage_threshold</i>	No	Determines the maximum number of allows services requests per configured interval. If the threshold is exceeded the module will warn the user. The minimum valid value 1. The value 0 is used to indicate "no limit".	No	0 (UNLIMITED)

**Example 6-28 UsageMonitoringModule**

This example configures the UsageMonitoringModule.

```
<Module>
  <Name>check_time</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.UsageMonitoringModule
  </Class-Name>
  <Param name="collection_interval" value="3600"/>
  <Param name="usage_threshold" value="0"/>
</Module>
```

## WindowsAdvancedAuthModule

`com.hp.ov.activator.mwfm.engine.module.umm.WindowsAdvancedAuthModule`

The module provides authentication and authorization functionality based on the underlying Windows Operating System authentication mechanism. It is only suitable for use on Windows.

The roles defined in the operating system and used by Service Activator must also be created through the User Management Interface. Only users which have this configuration are able to login to Service Activator. During installation of Service Activator the roles “admin” and “internal” are created for User Management. The “admin” role must be configured in the operating system to make it possible to enable the WindowsAdvancedAuthModule. However it is possible to use role mapping and hence create a different role for the System User in the underlying operating system.

Note that usernames are groups written as DOMAIN\USER and DOMAIN\GROUP. If the group or user is on the local system, it is written as USER or GROUP.

Name this module “authenticator” in the configuration.

### NOTE

The Windows operating system itself must be configured properly to allow Service Activator to perform this kind of authentication. See the discussion “Update Local Security Policies” on page 18 of the HP Service Activator Installation Guide for Windows.

### See Also

- “Roles, Privileges, and Authentication” in the *HP Service Activator - System Integrator’s Overview*
- “Configuring Authentication or Authorization” on page 68 in *HP Service Activator—Developing Plug-Ins and Compound Tasks*
- “Setting Roles” on page 29

**Table 6-34 WindowsAdvancedAuthModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<code>mwfm_remote_url</code>	Yes	Used to indicate from where to get the remote Workflow Manager service	No	None
<code>sleep_time</code>	No	The time between the internal role cache is cleared.	No	30 min
<code>eight_char_password</code>	No	If password should be truncated to 8 characters before authentication is done	No	false

**Table 6-34** WindowsAdvancedAuthModule Parameters (Continued)

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Reconfigurable</b>	<b>Default</b>
secure_user_name	No	Transform the user name to a valid value, cutting the user name from the first invalid character.	No	false

**Example 6-29**      **WindowsAdvancedAuthModule Code**

This example configures the WindowsAdvancedAuthModule with the workflow Manager service.

```
<Module>
  <Name>authenticator</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.umm.WindowsAdvancedAuthModule
  </Class-Name>
  <Param name="mwfm_remote_url" value="//localhost:2000/wfm"/>
</Module>
```

## WorkManagerModule

`com.hp.ov.activator.mwfm.engine.module.WorkManagerModule`

The Workflow Manager requires a work manager module to manage processing of work items. There is one class supplied to provide this functionality.

In addition, this module may be parameterized with a queue class. The queue specified affects how the work items are ordered. By default, this module uses the `SimpleEngineQueue`. This queue orders work items from the workflows in a strictly round-robin fashion.

Instead of the `SimpleEngineQueue`, you can choose the `PriorityEngineQueue` or `WeightedEngineQueue`, in which case work-items are processed with varying priority. A typical usage is to have most workflows executed at a neutral priority (0). If an important activity is requested that needs immediate processing, however, the workflow can have a high priority set.

When using the `PriorityEngineQueue` or the `WeightedEngineQueue`, the module looks for a case-packet variable with the name `PRIORITY` to determine the priority for the work items. Items of a higher priority are processed before items of a lower priority. If the `PRIORITY` case-packet variable is not found, the priority for the workflow is assumed to be 0. It is possible for the priority of a workflow to change during its life-time. This queue will recognize the new priority value.

**NOTE** The Workflow Manager requires a work manager module. By default, the Workflow Manager uses this module as its work manager and uses the `WeightedEngineQueue`.

**Table 6-35 WorkManager Parameters**

Parameter	Required	Description	Reconfigurable	Default
<code>queue_class</code>	No	Indicates which class implements the queue for ordering work items.	No	<code>com.hp.ov.activator.mwfm.engine.module.WeightedEngineQueue</code>

**Example 6-30 WorkManagerModule**

```
<Module>
  <Name>work_manager</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.WorkManagerModule
  </Class-Name>
  <Param name="queue_class"
    value="com.hp.ov.activator.mwfm.engine.module.PriorityEngineQueue"/>
</Module>
```

## XMLLogModule

`com.hp.ov.activator.mwfm.engine.module.XMLLogModule`

The Workflow Manager requires a module called `log_manager` to provide logging functionality. `XMLLogModule` is the preferred class for this purpose. `XMLLogModule` logs messages to files in the directory `$ACTIVATOR_VAR/log`.

`XMLLogModule` performs automatic rollover of log files after it has logged the configured number of messages.

Specify the module name as “`log_manager`.”

### See Also

- Appendix C, “Log Files,” on page 59 in *HP Service Activator— Introduction and Overview* for an example of log message format

**Table 6-36 XMLLogModule Parameters**

Parameter	Required	Description	Reconfigurable	Default
<code>log_max_entries</code>	No	The maximum number of log entries written to a log file before the log file is closed and a new one is created.	Yes	1000
<code>log_max_files</code>	No	The maximum number of log files which can exist. If the max number is reached the first file will be overwritten. The log module will always starts with creating log file with number 0. So the log module will rotate the log files over time. By setting this option the disc full situation can be avoided. If value is set to 0 no rotation is done.	No	0
<code>log_level</code>	No	The following <code>log_level</code> parameters set the type of information logged: ERROR - to record only error messages logged . WARNING - to record errors and warnings. INFORMATIVE - to record errors, warnings and some additional information. DEBUG - to get additional debugging information . DEBUG2 - to get even more detailed debugging information.	Yes	INFORMATIVE



**Table 6-36 XMLLogModule Parameters (Continued)**

<b>Parameter</b>	<b>Required</b>	<b>Description</b>	<b>Reconfigurable</b>	<b>Default</b>
<i>log_allow_statistics</i>	No	A value of "true" allows the logging of statistical data. This is distinct from the level of logging that is chosen.	No	"false"
<i>log_max_files</i>	No	The maximum number of log files that can exist. If the max number is reached the first file will be overwritten. The log module will always start with creating log file with the number 0. So the log module will rotate the log files over time. By setting this option you can avoid that the size of the log files grows indefinitely. If the value is set to 0 (default value) no log rotation is done.	No	0
<i>log_directory</i>	Yes	The location where the log files will be stored under the hostname folder.	No	VAR_HOME/log
<i>index</i>	No	This optional parameter determines whether log messages will be indexed for searching or not. A value of "true" will enable indexing, while "false" (default value) or any other value will disable it	Yes	false

**Example 6-31 XMLLogModule Code**

```
<Module>
  <Name>log_manager</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.XMLLogModule
  </Class-Name>
  <Param name="log_level" value="INFORMATIVE"/>
  <Param name="log_max_entries" value="1000"/>
  <Param name="log_allow_statistics" value="true"/>
</Module>
```

---

# 7

## Writing Custom Workflow Nodes

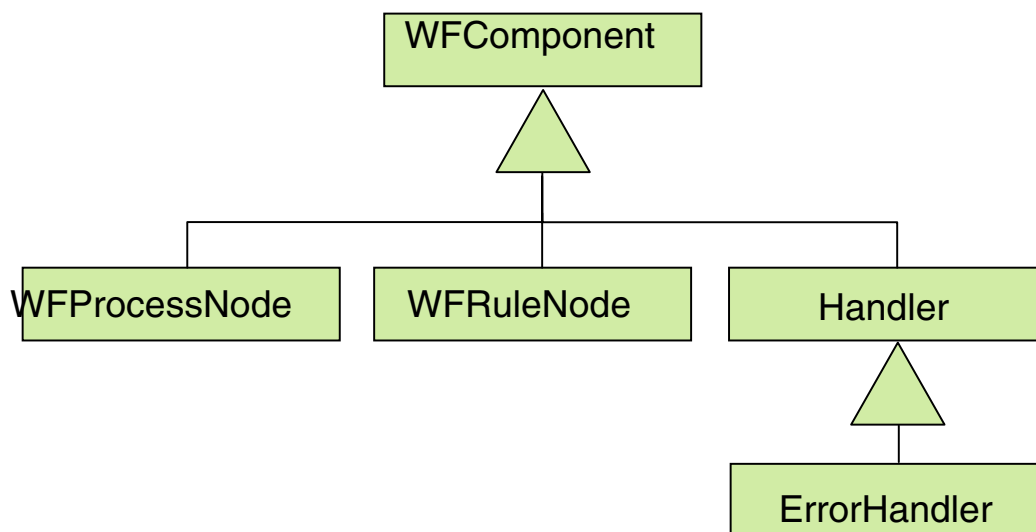
You may find it necessary to develop new workflow nodes to accomplish specific tasks in your business processes. This chapter provides conceptual information and instructions for writing new process nodes, rule nodes, and handlers.

## Understanding Workflow Nodes

Each workflow node and handler is implemented by a Java class that is derived from the `WFComponent` class.

Figure 7-1 is the inheritance hierarchy for `WFProcessNode`, `WFRuleNode`, `WFSwitch`, `WFComponent`, `Handler`, and `ErrorHandler`.

**Figure 7-1** Workflow Component Inheritance Hierarchy



All components shown in Figure 7-1 are subclasses of `WFComponent`.

There are five types of components, and each one is derived from a separate class:

Process nodes	<code>com.hp.ov.activator.mwfm.component.WFProcessNode</code>
Rule nodes	<code>com.hp.ov.activator.mwfm.component.WFRuleNode</code>
Switch nodes	<code>com.hp.ov.activator.mwfm.component.WFSwitch</code>
Error handler	<code>com.hp.ov.activator.mwfm.component.ErrorHandler</code>
End handler	<code>com.hp.ov.activator.mwfm.component.Handler</code>

**NOTE**

All of the shipped WF Nodes and Handlers are part of the `com.hp.ov.activator.mwfm.component.builtin` package. Your new nodes and handlers should use a different package name.

## Accessing Workflow Manager Capabilities: WFContext & WFManager

Workflow components (nodes and handlers) have access to two objects when they run: `WFContext` and `WFManager`.

The `com.hp.ov.activator.mwfm.component.WFContext` interface offers an efficient way to work with the case-packet, write to logs, interact with modules, and to obtain a reference to the `WFManager` remote interface.

The remote interface `com.hp.ov.activator.mwfm.WFManager` lets you interact with Service Activator Workflow Manager to start new jobs, get information on the status of these jobs, stop running jobs and pass values to a job that is paused waiting for input. The ability to pass values to the variables of a waiting job is an important feature that is discussed in detail later.

The base class of all Workflow Nodes contains a protected member variable, “context,” that provides access to the `WFContext` object. You obtain the interface to the `WFManager` through a context method.

The `WFComponent` class contains a series of helper methods for getting and setting case packet variables etc. These methods should be used to ensure consistent use of constant: and variable: plus a consistent behavior of the created nodes.

For details about the `WFManager`, `WFComponent` and `WFContext` classes see the *Javadocs*.

### Example Source Code for Nodes

The source code for a few example nodes are shipped with the product. All of the example nodes can be found in `$ACTIVATOR_OPT/examples/nodes`.

## Writing Custom Process Nodes

Process nodes are subclasses of `com.hp.ov.activator.mwfm.component.WFProcessNode`. They appear as:

```
public class Add extends WFProcessNode
```

These classes must declare a default constructor (with no parameters). When a workflow is started, one of these objects is instantiated for each node in the workflow. Thus, if a workflow has four Add nodes, then four objects of this class are instantiated.

They have two required public methods and one optional method:

- `init()` - required
- `nodeEntered()` - required
- `nodeExited()` - optional

### `init()` Method

This method is invoked on each instantiated node object in a workflow when the workflow is first started. It allows you to verify that all parameters have been declared and to initialize the component. This method should do as much parameter verification as possible. Any exceptions that are thrown during the `init()` method prohibit the workflow from actually starting.

The method is declared as follows:

```
public void init( HashMap config ) throws WFConfigException
```

The first statement in this method should send the configuration to the main class of the `WFComponent` using the following format:

```
super.init( config );
```

This way the `WFComponent` main class is properly initialized.

`config` is an object belonging to the `HashMap` class. It contains the parameters that are passed to the node. Each key is the name of the parameter and each value is the value as specified in the workflow definition.

If an error is detected during the `init` method, raise an exception belonging to the `WFConfigException` class in the following format:

```
throw new WFConfigException( "Some variable is needed" );
```

### `nodeEntered()` Method

This method is invoked when the workflow enters the corresponding node. Declare it in the following format:

```
public void nodeEntered() throws WFException
```

This is where you carry out the desired operations for your component.

If an error is detected, raise an exception belonging to the `WFException` class in the following format:

```
throw new WFException( "First param must be a variable" );
```

### `nodeExited()` Method

This optional method is invoked when the workflow exits the corresponding node. Declare it in the following format:

```
public void nodeExited() throws WFException
```

This is where you can carry out the completion of the operation for your component. Typically, this method is not needed.

### Example 7-1 Add - Example of a Process Node

This example shows the code for the Add node. The node receives a list of variables and constant values that it adds together. The result is stored with the first parameter, which must be a variable.

```

package com.hp.ov.activator.mwfm.component.builtin;

import java.util.*;
import java.text.*;
import com.hp.ov.activator.mwfm.component.*;
import com.hp.ov.activator.mwfm.engine.object.*;

public class Add extends WFProcessNode
{
    private String varToSet = null;
    private Vector attributes;

    public void init( HashMap config ) throws WFConfigurationException
    {
        super.init (config);
        attributes = new Vector();

        for ( int i=0 ; ; i++ ) {
            String str = (String) config.get( ObjectConstants.OPERAND + i);
            if ( str == null )
                break;

            if ( i == 0 ) {
                if ( str.startsWith(ObjectConstants.PREFIX_CONSTANT) ) {
                    throw new WFConfigurationException( MessageFormat.format("Parameter {0}
                        must be a variable.",
                            new Object[]{ObjectConstants.OPERAND+0} ) );
                }
                varToSet = str;
            }
            attributes.addElement( str );
        }

        if (attributes.isEmpty()) {
            throw new WFConfigurationException( MessageFormat.format( "The parameter {0}
                must be specified.",
                    new Object[] {ObjectConstants.OPERAND+0} ) );
        }
    }

    public void nodeEntered() throws WFException
    {
        double total = 0;
        int count = attributes.size();

        for ( int i=0 ; i < count ; i++ ) {
            String var = (String)attributes.get(i);
            // Fetch either the constant value or the
            // case-packet variable pointed to!
            String strVal = getStringValue(var);
            double val;

            try {
                val = (double)Double.parseDouble( strVal );
            } catch (Exception e) {

```

```
        throw new WFException( "The value specified is not numeric: " +
                                strVal );
    }
    total += val;
}

// if only one variable was given then we do a simple increment
if ( count == 1 )
    total++;

if ( context.getAttributeType( varToSet ).equals( "Integer" ) ) {
    Double d = new Double(total);

    // convert to an integer
    long longTotal = d.longValue();
    setValue(varToSet, "" + longTotal);
} else {
    setValue(varToSet, "" + total);
}

context.logDebug2( "new value for variable '" + varToSet + "' is " +
                    total );
}
}
```

#### Example 7-2 Use of `WFContext.requestUserInteraction()`

This is an example showing the use of the `WFContext.requestUserInteraction()` method within the `nodeEntered()` method of a process node. This is how a node can request the workflow to stop and wait for some values to be supplied by an external entity (either a human operator or a separate executable or workflow).

You should be aware that `WFContext.requestUserInteraction()` is non-blocking. The call returns after posting a request onto the given queue. After the `nodeEntered()` method returns, the Workflow Manager pauses the running workflow to wait for the requested input. After the user interacts with the workflow, the Workflow Manager executes the `nodeExited()` method of this node.

```
String[] vars = { "weekday", "startTime" };
String[] descriptions = {
    "What day to start the action",
    "Start time" };
boolean[] editable = { true, true };

int timeout = 0; // no timeout
String newQueue = "operator";
java.util.HashMap table = new HashMap();

// the days array will be used to give a pick-list for the weekday variable
String[] days = new String[7];
days[0]="Monday";
days[1]="Tuesday";
days[2]="Wednesday";
days[3]="Thursday";
days[4]="Friday";
days[5]="Saturday";
days[6]="Sunday";
table.put( vars[0], days );

context.requestUserInteraction( vars, descriptions, editable, table, newQueue,
    new Validator() {
        public java.lang.Object validate(java.util.HashMap
            requestedCasePacket)
```



```
throws WFInvalidCasePacketException {  
  
    if (vars[0]==null) {  
        throw new WFInvalidCasePacketException  
            ("Variable are mandatory and cannot be empty.");  
    }  
    if (validator != null)  
        return validator.validate (requestedCasePacket);  
    }  
},  
timeout );
```

## Writing Custom Rule Nodes

Custom rule nodes are subclasses of `WFRule`. They appear as:

```
public class Equal extends WFRule
```

They have two public methods that must be declared:

- `init()` Method
- `eval()` Method

### `init()` Method

In the same way as process nodes, this method is invoked when you start the flow. It lets you verify that all parameters have been declared and initializes the component. It appears as follows:

```
public void init( HashMap config ) throws WFConfigException
```

Send the configuration to the main class of the `WFComponent` with the following statement:

```
super.init( config );
```

This is to start the `WFComponent` main class correctly.

`config` is an object belonging to the `HashMap` class. Each key is the name of a parameter that has been passed to the component.

If an error is detected, throw an exception belonging to the `WFConfigException` class in the following format:

```
throw new WFConfigException( "Some variable is needed" );
```

### `eval()` Method

This method is invoked when the workflow enters the node. You declare it in the following format:

```
public boolean eval() throws WFException
```

It returns a `Boolean` value indicating whether the condition is true or false.

If an error is detected, throw an exception belonging to the `WFException` class in the following format:

```
throw new WFException( "First param must evaluate to a numeric value." );
```

### Example 7-3

#### Not – Example of a Rule Node

If you pass a variable, constant or string to the `Not` component, and the component evaluates it as a `NOT` in the `C` programming language (the variable or constant value is 0 or the string is empty), the returned value is “true.” Otherwise, it is “false.”

```
package com.hp.ov.activator.mwfm.component.builtin;  
  
import com.hp.ov.activator.mwfm.component.*;  
import java.util.*;  
  
public class Not extends WFRule  
{
```

```

// Operand's name
public static final String OP1    = "op1";
// The prefix to be a constant
public static final String CONSTANT = "constant:";

/** Method invoked when the workflow is started */
public void init (HashMap config) throws WFConfigException
{
    super.init (config);

    if (!config.containsKey (OP1))
        throw new WFConfigException (OP1 + " is mandatory parameter for " +
            "not nodes");
}

// The method to evaluate a rule.
public boolean eval() throws WFException
{
    String op1;
    long valor;

    op1 = (String) config.get (OP1);

    if (op1.startsWith(CONSTANT)){
        op1 = op1.substring(CONSTANT.length());
        try {
            valor = Long.parseLong(op1);
        } catch (Exception e) {
            throw new WFException("Constant seems not be a number");
        }
    }
    else
    {
        String str = context.getAttributeType (op1);
        if (str.compareTo("String")==0) {
            value = ((String) context.getAttribute (op1)).length();
        } else if (str.compareTo("Integer")==0) {
            try {
                value = ((Long)context.getAttribute (op1)).longValue();
                //Long.parseLong((String)
            } catch (NumberFormatException e) {
                throw new WFException("Integer not valid");
            }
        } else {
            throw new WFException("op1 is not a constant,string or integer");
        }
    }
    return valor!=0;
}
}

```

## Writing Custom Switch Nodes

Custom switch nodes are subclasses of `WFSwitch`. They appear as:

```
public class SwitchCase extends WFSwitch
```

They have two public methods that must be declared:

- `init()` Method
- `evalKey()` Method

### **init()** Method

In the same way as process nodes, this method is invoked when you start the flow. It lets you verify that all parameters have been declared and initializes the component. It appears as follows:

```
public void init( HashMap config ) throws WFConfigurationException
```

Send the configuration to the main class of the `WFComponent` with the following statement:

```
super.init( config );
```

This is to start the `WFComponent` main class correctly.

`config` is an object belonging to the `HashMap` class. Each key is the name of a parameter that has been passed to the component.

If an error is detected, throw an exception belonging to the `WFConfigurationException` class in the following format:

```
throw new WFConfigurationException( "Some variable is needed" );
```

### **evalKey()** Method

This method is invoked when the workflow enters the node. You declare it in the following format:

```
public string evalKey() throws WFException
```

It returns a `String` value indicating the value of the case branch.

If an error is detected, throw an exception belonging to the `WFException` class in the following format:

```
throw new WFException( "First param must evaluate to a numeric value." );
```

## Writing Error and End Handlers

Error handlers and end handlers are special workflow components that allow a workflow to perform some last-minute functions before the workflow finishes running. There are only a few built-in handlers supplied; however, you can write your own.

The two types of handlers have only a single required method called `execute()`. This method is invoked when the workflow is being terminated. Since the handlers are also workflow components they have an `init()` method just like process and rule nodes. Thus, they can be parameterized in the same fashion as other workflow nodes. A handler that is intended to be an error handler must extend the `ErrorHandler` class. A handler intended to be used as an end handler need only extend the `Handler` class.

It is possible to write a handler suitable for either purpose, in which case it should implement the `ErrorHandler` interface.

If the handler is used only as an end handler, the `setException()` method is not called before the `execute()` method. The built-in handlers are written in this way:

### Example 7-4

#### MyErrorHandler – Example of an Error Handler

This is an example of a handler that puts a message on the standard output of the Workflow Manager.

```
import java.util.*;

import com.hp.ov.activator.mwfm.component.*;
import com.hp.ov.activator.mwfm.engine.object.*;

// Invoked when the workflow encounters an error
public class MyErrorHandler extends ErrorHandler {

    public void execute(){
        System.out.println("**** Exception captured by MyErrorHandler ****");
    }
}
```

## Deploying Workflow Nodes and Handlers

If you write custom workflow components, the Workflow Manager must be able to find them. The Workflow Manager uses the standard Java mechanism for finding such classes, the “classpath”.

Only jar files are supported so the new classes must be packed into jar file(s)..

The jar files must be placed in the directory `$JBOSS_EAR_LIB`. If a jar file is placed in this directory Service Activator must be restarted to find the jar file.

---

### NOTE

During development you may change the implementation of your new nodes. The Workflow Manager will not notice an updated jar file after Service Activator has been started. If you change your jar files, then you must stop and restart Service Activator to pick-up the new implementation.

---

## Using Custom Nodes and Handlers in Designer



You can make your new components available when using the designer to create workflows. Each node must have a **Component Descriptor** file.

To be recognized by the WF Designer, all of the Node Descriptor files must be placed under the directory `$ACTIVATOR_ETC/designer/nodes`. In this directory you see the directory “builtin.” This directory contains the descriptors for all of the built-in nodes. Your node descriptors may be placed in this directory, or they may be placed in other directories. The designer shows the list of available nodes in tabs on the left hand side of the workspace. For each directory that the designer finds, it creates a new tab. These tabs are built when the designer is started. If you add new node descriptors, then you need to restart the designer.

All of the Handler Descriptor files must be placed under the directory `$ACTIVATOR_ETC/designer/handlers/end` or `$ACTIVATOR_ETC/designer/handlers/error`. These handler directories do not support subdirectories as in the case of nodes.

### Component Descriptor Vocabulary

These files are, as you might expect, written in a special XML vocabulary. The descriptor specifies:

- the name of the node or handler
- the name of the java class that implements it
- an image to display on the node button in place of the  or the  icon
- a description of the component behavior
- the type of the component (process node, rule node, end handler, or error handler)
- a list of parameters supported by the component, and the details of each parameter

The XML vocabulary is defined in `workflowComponent.dtd`. This DTD is found in the `nodes/builtin` directory.

## Example 7-5 Node Descriptor Example

This example shows the node descriptor for the built-in node `StartJob`. Most of the content is self explanatory except for the attributes on the parameters. They are described below.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE WorkflowComponent SYSTEM 'workflowComponent.dtd'>
<!-- Copyright (c) 2000-2002 Hewlett-Packard Company. All Rights Reserved -->

<WorkflowComponent>
  <Name>StartJob</Name>
  <NodeDescription>Begins execution of a new workflows.</NodeDescription>
  <ClassName>com.hp.ov.activator.mwfm.component.builtin.StartJob</ClassName>
  <Type>ProcessNode</Type>
  <DisablePersistence>>false</DisablePersistence>
  <Params>
    <Param Required="true" Constant="true">
      <Name>workflow_name</Name>
      <Description>
        The name of the workflow to start. May be a variable or a constant.
      </Description>
    </Param>

    <Param Multiple="true">
      <Name>variable</Name>
      <Description>
        Case-packet variables that are to be passed to initialize variables
        in the new workflow being started.
      </Description>
    </Param>

    <Param Multiple="true" Related_Param="variable">
      <Name>destination</Name>
      <Description>
        The name of the case-packet variable to initialize in the new
        workflow. By default the variable of the same name is initialized.
      </Description>
    </Param>
  </Params>
</WorkflowComponent>
```

First, notice the parameter `workflow_name`. The attributes indicate that this parameter is required. Also you see the attribute `Constant` is set to `true`. This attribute indicates that the parameter, by default, accepts the name of a case-packet variable. Thus, if the user indicates that the parameter is a constant, then it should have the phrase “constant:” prepended.



Next, notice the parameter `variable`. The attributes indicate that this parameter may appear multiple times in the parameter list for this node. The designer automatically appends an incrementing number to the parameter name to distinguish and order these parameters.

Finally, notice the parameter “`destination`”. This is also a multiple parameter, but you can see that it is supposed to be related to the “`variable`” parameter. This means that a parameter with the name “`destination5`” is related to a parameter with the name “`variable5`”.

### DTD Quick Reference.

The following table describes the XML vocabulary that is used for the Node Descriptors. Your Descriptors must reference this DTD for Designer to be able to process them. See the StartJob example above for details about how the DTD specification is made.

**<WorkflowComponent>** - the root tag of the XML specification

- **<Name>** - tag declaring the name of this node
- **<NodeDescription>** - tag providing a complete description of the behavior of the node and any inter-parameter dependencies. The user of the designer can see this description by asking for help on the node.
- **<ClassName>** - tag specifying the java class (including package name) that implements this node
- **<Image>** - tag indicating an icon to show in place of the default  or  that is displayed on the button for this node. You may use the relative path to an icon in one of the jars that is in the classpath of the Workflow Designer, or you may specify a file path relative to the `<ImagesDirectory>` configured in the `designer.xml` file.
- **<Type>** - tag indicating whether this is a `ProcessNode`, `RuleNode`, `EndHandler`, or `ErrorHandler`
- **<DisablePersistence>** - tag indicating wheter this node should do persistence or not after execution of the node.
- **<Params>** - tag defining the list of supported parameters
  - **<Param>** - tag defining a single parameter
    - **Required** - *optional* attribute indicating whether the parameter is required for this node (default=**false**).
    - **Constant** - *optional* attribute indicating that if the user specifies that the parameter value is a constant, then the designer should prepend “constant:” to the value (default=**false**).
    - **Variable** - *optional* attribute indicating that if the user specifies that the parameter value is a variable, then the designer should prepend “variable:” to the value (default=**false**). None of the built-in nodes currently use this setting.
    - **Multiple** - *optional* attribute indicating whether this parameter should be treated as repeatable parameter with an incrementing number automatically appended to the name (default=**false**).
    - **Related\_Param** - *optional* attribute indicating what other parameter this one is related to. This is only meaningful if both of them are “multiple” parameters.
    - **<Name>** - tag specifying the name of the parameter
    - **<Description>** - tag providing a description of the parameter. This appears in the parameter dialog when the user is editing the parameter value.



---

## **8 Writing New Workflow Modules**

The Workflow Manager comes with a catalog of workflow modules. You may find it necessary to develop new workflow modules to communicate with new external systems. This chapter provides conceptual information and instructions for writing new modules.

---

## Writing New Workflow Manager Modules

Service Activator ships with many built-in Workflow Manager modules as described in the previous section; however, it is possible to write new Workflow Manager modules to replace or enhance existing functionality. These include:

- Authenticator modules
- Queue hook modules
- Sender modules
- Alarm modules
- Listener modules
- Activation modules

You can even create new modules with special functionality to support the behavior of new nodes that you write.

Some modules are used by the workflow engine itself, other modules are used by certain nodes. Examples of Workflow Manager modules that the Workflow Manager uses include a logging module or an authenticator module. A module that supports specific node behavior is the `SocketSenderModule`, which is used by the `SendMessage` node. Workflow nodes that need to interact with a database use the `DatabaseModule`. In either case, these Workflow Manager modules must support the interface that is unique to the functionality they provide.

### Example Source Code for Modules

The source code for a few example modules is shipped with the product. All of the example modules can be found in `$ACTIVATOR_OPT/examples/modules`.

## Implementation of Modules

Each Workflow Manager module is implemented by a Java class. The class must extend `WorkflowManagerModule` or must extend one of the existing classes that already extends `WorkflowManagerModule`. In some cases the module must also implement a specific Java interface (see the details provided in this chapter for specific interfaces or base classes that might be required for the particular type of module you are writing).

When the Workflow Manager starts up, it creates an instance of each Java class configured as a module in the `mwfm.xml` file. It then invokes setter methods for each of the parameters configured for the module. The naming of these setter methods adheres to the conventions for a JavaBean. For example, if the parameter name is `maxPriority`, then the name of the method is `setMaxPriority`. If there is not a setter method for the specified parameter, a warning message is logged.

After invoking setter methods for each configured parameter, the Workflow Manager invokes the `init()` method of the module. The `init()` method is passed a `HashMap` that contains all of the configuration parameters that are declared in the `mwfm.xml` file for that module. During the `init` method, the module checks for any required configuration parameters and saves the value of the parameters for later use by the module.

When the `init` method is called the module is in the suspended state. The module should not start to perform any normal activity like start of jobs or interaction with jobs until the method `resume` is called.

Other methods of that class are used at the appropriate time. For example, if the class is a new authenticator module, then `isUserPasswordPairValid()` is invoked each time a user attempts to log on.

### **init Method**

This method is common to all Workflow Manager modules and allows you to initialize the module, verify the parameters, and so on. As noted above, the preferred mechanism to set configurable parameters is via the setter methods. The `init()` method is still valuable when the module needs to enforce any inter-parameter prerequisites or needs to throw an exception if a parameter is missing. The module is in suspended state when this method is called. So no normal operation should be done in the `init` method. In case it is not possible to get a database connection then an `WFConnectivityException` should be thrown.

```
void init(HashMap params) throws WFConfigException, WFException
```

### **shutdown Method**

Some modules need to perform a special shutdown procedure when the workflow engine is being shutdown in a graceful fashion. If the module does not need special processing in this case, it does not need to implement this method.

```
void shutdown()
```

### **isActive Method**

When the engine attempts to shutdown gracefully, it first calls the `isActive()` method of each registered module. It does not shutdown until all the modules have responded with a value of “false”. The implementation of this method in the base class always returns “false”.

```
boolean isActive()
```

### **removeJob Method**

Some modules keep track of running jobs. When a job is terminated prematurely (for example, by the operator killing the job), the engine needs to tell these modules that they should forget about that particular job. If the module does not track running jobs, it does not need to implement this method.

```
void removeJob( long jobId )
```

### **reconfigure Method**

This method is called when a reload configuration is issued. The params `HashMap` include all the new configuration elements. If the module needs to support reconfiguration it must overwrite this method and handle the config changes. If the module does not support reconfiguration it does not need to implement this method.

```
void reconfigure( HashMap params )
```

### **suspend Method**

When a cluster node is suspended either due to the loss of database connectivity or an operator suspend, the Workflow Manager calls the `suspend()` method of each registered module. When the system is suspended, each module will have its own implementation method to perform a suitable operation. For example, the socket listener module will suspend itself and stop processing requests when the system is suspended due to loss of database connectivity. If the module does not need special processing in this case, it does not need to implement this method.

```
void suspend()
```

### **resume Method**

When a cluster node is resumed after the database connectivity is restored or an operator resume, the Workflow Manager calls the `resume()` method of each registered module. When the system is resumed, each module will have its own implementation method to perform a suitable operation. For example, the `SocketListener` Module will notify all waiting threads to continue processing requests which were stopped when the system was suspended, or resumed due to restoration of database connectivity. If the module gets database connectivity problems during the resume the method `setStateNotificationFailure` must be called with the parameter “true” and the stop further processing until resume is called again. If the module does not need special processing in this case, it does not need to implement this method.

```
void resume()
```

### **locked Method**

When a cluster node is locked from the operator UI, the Workflow Manager calls the `locked()` method of each registered module. When the system is locked, each module will have its own implementation method to perform a suitable operation. For example, the Scheduler Module will stop scheduling jobs when the system is locked. If the module does not need special processing in this case, it does not need to implement this method.

```
void locked()
```

### **unlocked Method**

When a cluster node is unlocked from the operator UI, the Workflow Manager calls the `unlocked()` method of each registered module. When the system is unlocked, each module will have its own implementation method to perform a suitable operation. For example, the Scheduler Module will restart scheduling of jobs when the system is unlocked. If the module does not need special processing in this case, it does not need to implement this method.

```
void unlocked()
```

### **nodeDown Method**

When the online state of other cluster nodes change from online to offline, the KeepAlive Module that monitors the state, calls the `nodeUp()` method of each registered module. When another node comes online, each module will have its own implementation method to perform a suitable operation. If the module does not need special processing in this case, it does not need to implement this method.

```
void nodeDown(ClusterNodeBean node)
```

### **nodeUp Method**

When the online state of other cluster nodes change from offline to online, the KeepAlive Module that monitors the state, calls the `nodeDown()` method of each registered module. When another node goes offline, each module will have its own implementation method to perform a suitable operation. For example, when a node goes offline, the Scheduler Module tries to become the master scheduler for the cluster. If the module does not need special processing in this case, it does not need to implement this method.

```
void nodeUp(ClusterNodeBean node)
```

### **nodeLocked Method**

When the lock state of other cluster nodes change from unlocked to locked, the KeepAlive Module that monitors the state, calls the `nodeLocked()` method of each registered module. When a node is locked, each module will have its own implementation method to perform a suitable operation. For example, when a node is locked, the Scheduler Module tries to become the master scheduler for the cluster.

```
void nodeLocked(ClusterNodeBean node)
```

**nodeUnlocked Method**

When the lock state of other cluster nodes change from locked to unlocked, the KeepAlive Module that monitors the state, calls the `nodeUnlocked()` method of each registered module. When a node is unlocked, each module will have its own implementation method to perform a suitable operation. If the module does not need special processing in this case, it does not need to implement this method.

```
void nodeUnlocked(ClusterNodeBean node)
```

**nodeSuspended Method**

When the suspend state of other cluster nodes change from resumed to suspended due to an operator suspend, the KeepAlive Module that monitors the state, calls the `nodeSuspended()` method of each registered module. When a node is suspended, each module will have its own implementation method to perform a suitable operation. For example, when a node is suspended, the Scheduler Module tries to become the master scheduler for the cluster. If the module does not need special processing in this case, it does not need to implement this method.

```
void nodeSuspended(ClusterNodeBean node)
```

**nodeResumed Method**

When the suspend state of other cluster nodes change from suspended to resumed due to an operator resume, the KeepAlive Module that monitors the state, calls the `nodeResumed()` method of each registered module. When a node is resumed, each module will have its own implementation method to perform a suitable operation. If the module does not need special processing in this case, it does not need to implement this method.

```
void nodeResumed(ClusterNodeBean node)
```

**takeover Method**

When jobs have been successfully takeover by one cluster node then the modules are notified by this method is called. If the module does not need special processing in this case, it does not need to implement this method.

```
void takeover(ClusterNodeBean node)
```

**refresh Method**

When a cluster node takes over the jobs that were being executed in a failed node, the KeepAlive Module calls the `refresh()` method of each registered module. When a node is taken over, each module will have its own implementation method to perform a suitable operation. The `isBeforeTakeOver` flag is used by the module if it wishes to perform any tasks before and after a failover process. The `refresh()` method is invoked on sync module before and after failover process. For example, the Sync Module, before failover updates the information from the database as to which parent workflows have spawned which child workflows, so that they can synchronize with each other. If the module does not need special processing in this case, it does not need to implement this method.

```
void refresh(String hostName, boolean isBeforeTakeOver)
```

## discard Method

When a cluster node is suspended due to loss of database connectivity, it resumes operation after the restoration of database connectivity. If the node determines that it is being taken over by another node in the cluster, it waits till the completion of the take over process, and the the KeepAlive Module calls the discard() method of each registered module. After a node is taken over, each module will have its own implementation method to perform a suitable operation. For example, the Sync Module cleans up the information related to parent and child workflows that are waiting to synchronize with each other. If the module does not need special processing in this case, it does not need to implement this method.

```
void discard()
```

## Master-Slave

The master-slave approach helps modules to ensure that the same module is started on all the nodes in a cluster, even though the behaviour is different depending on their state; either master or a slave.

In a distributed environment only one module can be a master. The modules with the same name running on other cluster nodes will automatically become a slave. The master-slave behavior can be changed when the node in which the module is a master goes down or suspended or locked and another node takes over. The node which takes over the job will update the state of the failed node to the slave and then update its state to the master.

The master-slave approach is best suited to handle situations such as, a possible conflict when the same module is running on all cluster nodes tries to execute the same task due to lack of communication between the modules. Each module running on a cluster node and is using this concept will either be in master or slave state, and the modules function based on their state. The master-slave concept is very specific to a distributed environment and insignificant in a standalone environment.

Any module running on a cluster node can implement this concept.

The following section explains the sequences to be followed by each module to implement this concept.

### Step 1

Each module must overwrite `init` method of its super class. This is the starting point for the module and will be invoked by mwfm engine. Once started, it has to create a new entry in modules table with `MasterSlaveState` as '0' (slave) as default, if no entry exists for this.. To do this, just call `createMasterSlaveState()` of its super class from this method.

### Step 2

Each module must overwrite `resume` method of its super class and this method is invoked by the KeepAlive module. When this method is invoked, it should try to become a master. To do this, just call `becomeMaster (null)` of its super class. If the module becomes a master, the mehtod will return true, or else false.

### Step 3

Each module must overwrite `nodeDown` method of its super class and this method is invoked by the `KeepAlive` module to inform the node down status of other nodes in the cluster system. When this method is invoked, it should try to become a master (take over), if the module running on the failed node is a master. To do this, just call `becomeMaster ()` of its super class and pass the name of the node which failed as an argument to this function. If the module becomes a master, the method will return true, or else false.

### Step 4

Each module must overwrite `nodeSuspended` method of its super class and this method is invoked by the `KeepAlive` module to inform the suspended status of other nodes in the cluster system. When this method is invoked, it should try to become a master (take over), if the module running on the suspended node is a master. To do this, just call `becomeMaster ()` of its super class, and pass the name of the node which is suspended as an argument to this function. If the module becomes a master, the method will return true, or else false.

### Step 5

Each module must overwrite `nodeLocked` method of its super class and this method is invoked by the `KeepAlive` module to inform the lock status of other nodes in the cluster system. When this method is invoked, it should try to become a master (take over), in case the module running on the locked node is a master. To do this, just call `becomeMaster ()` of its super class and pass the name of the node which is locked as an argument to this function. If the module becomes a master, the method will return true, or else false.

### Step 6

Each module must overwrite `unlocked` method of its super class and this method is invoked by the `KeepAlive` module to inform about the unlock status (the node is unlocked from lock state) of the same node. When this method is invoked, it should try to become a master (take over), to find out if any other master already exists. To do this, just call `becomeMaster (null)` of its super class. If the module becomes a master, the method will return true, or else false.



## Writing New Authenticator Module

You can supply your own authenticator module to use instead of one of those shipped with Service Activator. To function as an authenticator, the module must extend the abstract `com.hp.ov.activator.mwfm.engine.module.umm.AdvanceAuthModule` class.

```
AdvancedAuthModule class extends AuthModule class and implements RoleMappingSupport
and UserManagementManager interfaces
com.hp.ov.activator.mwfm.engine.module.AuthModule
com.hp.ov.activator.mwfm.engine.module.RoleMappingSupport
com.hp.ov.activator.mwfm.UserManagementManager
```

### AuthModule methods

The `AuthModule` class has a number of abstract methods which are implemented by the `AdvancedAuthModule` class.

### AdvancedAuthModule methods

The `AdvancedAuthModule` class has one abstract method which must be implemented.

```
boolean authenticate(String username, String password) throws AuthException
```

This method is called when an user attempts to log on to the system. The method verifies that the user is allowed to use Service Activator, and that the supplied password is appropriate for that user. It should return “true” to allow the user to log on, or “false” if not. `AuthException` should only be thrown if authentication could not be carried out.

### Role Mapping

Authenticators should also support the functionality known as role mapping. This allows workflow definitions and inventory JavaBeans to be written using generic role names that might be suitable anywhere. Then role mapping can be used to map these generic roles to real roles that are meaningful in a particular customer environment.

### Role Mapping Interface

To support role mapping, an authentication module must also implement an additional Java interface, `com.hp.ov.activator.mwfm.engine.module.RoleMappingSupport`, and a single method.

```
void setRoleMappings(RoleMapping roleMapping);
```

This method is called after `init()` to set the role mappings that the authenticator should recognize. The `RoleMapping` class has one important method that the authenticator can use to retrieve a list of roles that are mapped from a generic role.

```
String[] RoleMapping.getMappings( String role );
```

### User Management Interface

To support The User Management Interface, the authenticator module must also implement an additional Java interface, `com.hp.ov.activator.mwfm.engine.UserManagementManger`.

Almost all the methods in the `UserManagementManager` (UMM) interface are implemented in a default way in the `AdvancedAuthModule` class and data handled by these methods are saved in system database.

The UMM methods can be divided into two categories; methods related to user and team configuration and the rest which is handling roles, inventory trees, inventory filters and searches. The first category of methods has a dummy implementation in the `AdvancedAuthModule` and is fully implemented by the `DatabaseAdvancedAuth` module. The methods related to team configuration cannot be overwritten by a new authenticator module where the user part can be implemented in a different way when creating a new authenticator module. The role methods which are implemented in the `AdvancedAuthModule` should not be overwritten as these methods are used to configure which roles are known by Service Activator and the relation from roles to inventory trees, branch and operation types, filters and advanced search. However it is possible to extend the role methods to also create the roles in an additional system.

Two methods must always be implemented to support the User Management interface when creating a new authenticator module:

```
boolean isDBAuth() throws RemoteException;
```

This method must return a boolean to indicate if the user dummy methods are implemented in a meaningful way or not. If implemented then the UMM user interface will present the user information too.

```
java.lang.String[] getUserRoles(String username)
throws RemoteException, WfConnectivityException, WfDBException;
```

This method must return the list of roles the provided user has

The following user methods can be re-implemented. For a description of how to implement the methods please refer to Javadocs.

```
public void copyUser( String adminLogin, String adminPassword, String
originalUserName, String userName, String password, String userDescription, String
userRealName, String companyName, boolean restrictedUser, boolean firstTimeLogin,
boolean neverExpire, boolean enable) throws RemoteException,
WfConnectivityException, WfDBException;
```

```
public void createUser( String adminLogin, String adminPassword, String userName,
String password, String userDescription, String userRealName, String[] roleNames,
String companyName, boolean superUser, boolean restrictedUser, boolean
firstTimeLogin, boolean neverExpire, String teamName, boolean isTeamAdmin, boolean
enable) throws RemoteException, WfConnectivityException, WfDBException;
```

```
public void updateUser( String adminLogin, String adminPassword, String userName,
String newUserRealName, String password, String userDescription, String userRealName,
String companyName, boolean superUser, boolean restrictedUser, boolean
firstTimeLogin, boolean neverExpire, boolean enable, String[] roleNames, String
teamName, boolean isTeamAdmin) throws RemoteException, WfConnectivityException,
WfDBException;
```

```
public void dropUser( String adminLogin, String adminPassword, String userName)
throws RemoteException, WfConnectivityException, WfDBException;
```

```
public void assignUserRoles( String adminLogin, String adminPassword, String
userName, String[] roleNames) throws RemoteException, WfConnectivityException,
WfDBException;
```

```
public UserInfo[] getAllUsers( String adminLogin, String adminPassword) throws
RemoteException, WfConnectivityException, WfDBException;
```

```
public UserInfo getUser( String adminLogin, String adminPassword, String username)
throws RemoteException, WfConnectivityException, WfDBException;
```

```
public InvalidLoginAttempt[] getUserInvalidLoginAttempts(String userName) throws
RemoteException, WfConnectivityException, WfDBException;
```

```
public boolean isUserSuperUser( String userName) throws RemoteException,
WFConnectivityException, WFDBException;

public boolean isUserRestricted (String userName) throws RemoteException,
WFConnectivityException, WFDBException;

public void assignUserRoles( String adminLogin, String adminPassword, String
userName, String[] roleNames) throws RemoteException, WFConnectivityException,
WFDBException;

public String[] getUserRolesExt(String userName) throws RemoteException;

public UserInfo[] getRoleUsers( String adminLogin, String adminPassword, String
roleName) throws RemoteException, WFConnectivityException, WFDBException;

public void assignRoleUsers( String adminLogin, String adminPassword, String
roleName, String[] userNames) throws RemoteException, WFConnectivityException,
WFDBException;

public boolean changeUserPassword(String userName, String oldPasswd, String
newPasswd) throws RemoteException, WFConnectivityException, WFDBException,
AdvancedAuthModuleException;

public void updateUserAuthenticated(String userName, String password) throws
RemoteException, WFConnectivityException, WFDBException;

public void changePasswordFirstTimeLogin( String adminLogin, String adminPassword,
String newPassword) throws RemoteException, WFConnectivityException, WFDBException,
AdvancedAuthModuleException;

public void disableUser( String adminLogin, String adminPassword, String userName)
throws RemoteException, WFConnectivityException, WFDBException;

public void enableUser( String adminLogin, String adminPassword, String userName)
throws RemoteException, WFConnectivityException, WFDBException;

public String getExpiryAlertDays(String adminLogin, String adminPassword) throws
RemoteException;

public boolean checkFirstTimeLoginOuter(String userName, String password) throws
RemoteException;
```

## Writing New Queue Hook

Queue hooks are invoked whenever a new message arrives in either the request or message queues. One Queue hook module (see “LogSearchModule” on page 403) is supplied with the product that allows you to configure the system to send e-mail when new messages arrive on various queues. New queue hook modules that you might write could be used for other purposes such as to page an administrator when messages arrive on a special queue, or to inform another application that it has a request that it should process.

When a Queue hook module is configured in the `mwfm.xml` file, it must be given a module name according to a special convention. Each queue hook module must be given the name “hookN” where N is a number indicating the order in which the modules are informed of new messages. Thus, if there is only one hook module configured then it must be given the name “hook0.” A second hook module would be given the name “hook1,” and so on.

---

### NOTE

This discussion of queue hook naming refers to the module name that the module is configured to have in the `mwfm.xml` file, NOT to the class name that implements the module.

---

A new queue hook must extend the `WorkflowManagerModule` class and implement the interface, `com.hp.ov.activator.mwfm.engine.module.QueueHook`. The `QueueHookAdapter` class is provided as a convenience.

### QueueHook Methods

These are the methods that may be implemented to create a `QueueHook`.

```
void newAsynchronousMessage(MessageDescriptor md)
```

Invoked when a new message arrives on a message queue.

```
void newSynchronousMessage(String name, JobDescriptor jd)
```

Invoked when a new request arrives on a request queue.

### Example 8-1

#### QueueHook example

The source for the `MailHook` is provided as an example of a `QueueHook`.

See `$(ACTIVATOR_OPT)/examples/modules/MailHook.java`

## Writing New Sender Module

A sender module is invoked from the `SendMessage` workflow node. The `SocketSenderModule` is an example of such a module. You might want to provide a sender module to send a message by a mechanism other than TCP sockets. A sender module must implement the interface, `com.hp.ov.activator.mwfm.engine.module.SenderModule`, and a single method.

### SenderModule Methods

This method is invoked by the `SendMessage` workflow node to send a message.

```
void sendMessage( byte[] msg ) throws IOException;
```

## Writing New Message Module

A message module is invoked to send an event (or an alarm) to a Fault Management system such as OpenView Operations. The `OVOMessageModule` is an example of such a module. You might want to provide your own message module to send a message to a different fault management system. A message module must implement the interface, `com.hp.ov.activator.mwfm.engine.module.MessageModule`, and a single method.

### MessageModule Methods

Invoked by the `SendAlarm` workflow node to send an alarm.

```
void sendMessage( String msg, HashMap params ) throws IOException
```

The `params` argument contains a list of the name-value pairs that is used by the message module to parameterize the alarm. These parameter names that are meaningful are dependent upon the implementation of the alarm module. In case of `OVOMessageModule`, these are names like: `severity` and `msg_grp`.

## Deploying Workflow Manager Modules

If you write custom workflow modules, the Workflow Manager must be able to find them. The Workflow Manager uses the standard Java mechanism for finding such classes, the “classpath”.

Only jar files are supported so the new classes must be packed into jar file(s)..

The jar files must be placed in the directory `$JBOSS_EAR_LIB`. If a jar file is placed in this directory Service Activator must be restarted to find the jar file.

---

### NOTE

During development you may change the implementation of your new modules. The Workflow Manager will not notice an updated jar file after Service Activator has be started. If you change your jar files, then you must stop and restart Service Activator to pick-up the new implementation.





---

# 9

## Writing Workflow Manager Clients

You can write Java programs to interact with the Workflow Manager. These client programs can start workflows, interact with running jobs, examine message queues; most anything that can be done from the operator GUI. This chapter describes how to create such a program.

---

## Writing Workflow Manager External Interface Clients

This section describes the external RMI interface used to access the Workflow Manager, and provides some example programs that use the RMI interface. The details below assume that you are writing a Java client program. The RMI interface provides methods to perform the following operations:

- Authenticate a user so he or she can gain access to the Workflow Manager.
- Determine whether the user has a specific role (or set of roles).
- Determine whether the user is considered an administrator.
- Obtain a list of workflows that the user can start.
- Obtain the description of a workflow.
- Start a new job (an instance of a workflow).
- Obtain a list of the running jobs.
- Obtain details about a running job.
- Kill a running job.
- Set case-packet variables of jobs that are waiting for input.
- Obtain a list of the currently posted messages.
- Obtain a list of the currently posted requests.
- Set and get roles.
- Get valid role names from the authentication module.
- Get case packet information about a single job.
- Change the sorting of jobs and messages.
- Schedule a workflow. Query, modify or delete scheduled jobs.

To see the complete interface and a description of all the available methods, refer to the *Javadoc* for the Java interface `com.hp.ov.activator.mwfm.WFManager`.

There are other related classes in the same package. The rest of this section does not discuss the details of the interface, but discusses what is generally needed to use the interface.

---

## Creating a Workflow Manager Client

To interact with the Workflow Manager, you must first obtain a remote reference to the Workflow Manager authenticator. To do this, perform a naming lookup on the host where the Workflow Manager is running. For example:

```
WFAuthenticator wfauth = (WFAuthenticator)Naming.lookup( "//localhost:2000/wfm" );
```

This example specifies both the host name and the port in the `Naming.lookup`. The host is any reachable host in the network; the port is the one that you configured for the Workflow Manager to listen on (in the `mwfm.xml` file for the running Workflow Manager that you wish to connect to).

After you obtain a reference to the workflow authenticator, obtain a reference to the Workflow Manager itself by supplying a valid user name and password; you must be authenticated by the Workflow Manager.

```
WFManager wfm = (WFManager)wfauth.login( username, password );
```

If the supplied user and password are appropriate, then a `WFManager` object is returned. This object provides all of the functionality for gaining access to the Workflow Manager. You will have all of the restrictions according to the user by which you logged in; that is, you will be able to start, stop and interact with workflows according to the role(s) assigned to your user.

## Examples

Here are some example programs that demonstrating a few of the methods provided for interacting with the Workflow Manager.

### Example 9-1 GetJobStatus

This client gets the status of the job whose identifier corresponds to the job identifier that you passed.

```
package com.hp.ov.activator.mwfm.client;

import java.rmi.*;
import java.rmi.registry.*;
import java.util.*;
import java.io.*;

import com.hp.ov.activator.mwfm.*;

/**
 * A sample program to get the status of a job.
 *
 * @version $Revision: 2 $
 */
public class GetJobStatus
{
    public static void main (String[] args) throws Exception
    {
        if (args.length != 3) {
            System.out.println ("Usage: GetJobStatus <username> <password> <job-id>");
            System.exit (1);
        }

        WFAuthenticator wfauth = (WFAuthenticator) Naming.lookup ("//:2000/wfm");
        WFManager wfm = (WFManager) wfauth.login (args[0], args[1]);
        if (wfm == null) {
            System.err.println ("username/password incorrect");
            System.exit (2);
        }

        long l = Long.valueOf (args[2]).longValue();
        System.out.println ("STATUS for job #" + l + ": " + wfm.getJobStatus(l));
    }
}
```

**Example 9-2 SendCasePacket**

This example shows how to send values to case-packet variables of a workflow waiting for external input.

```
package com.hp.ov.activator.mwfm.client;

import java.rmi.*;
import java.rmi.registry.*;
import java.util.*;
import java.io.*;

import com.hp.ov.activator.mwfm.*;

/**
 * A sample program to send case-packet variables
 * to a pending process.
 * <p>
 * The case-packet is received from the standard input
 * in an attribute=value fashion.
 *
 * @version $Revision: 3 $
 */
public class SendCasePacket
{
    public static void main (String[] args) throws Exception
    {
        if (args.length != 4) {
            System.out.println ("Usage: SendCasePacket <username> <password> <queue>
                                <jobId>");
            System.exit (1);
        }

        WFAuthenticator wfauth = (WFAuthenticator) Naming.lookup ("//:2000/wfm");
        WFManager wfm = (WFManager) wfauth.login (args[0], args[1]);
        if (wfm == null) {
            System.err.println ("username/password incorrect");
            System.exit (2);
        }

        HashMap hash = new HashMap();
        BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
        String str;

        // build a HashMap of name/value pairs
        while ((str = br.readLine()) != null && !str.trim().equals("")) {
            StringTokenizer strToken = new StringTokenizer (str, "=");
            hash.put (strToken.nextToken(), strToken.nextToken());
        }
        System.out.println ("RESPONSE: " + wfm.sendCasePacket (args[2], Long.valueOf
                                                                (args[3]).longValue(), hash));
    }
}
```



---

## **E** **Configuring Service Activator to Use Secure Socket Layer (SSL) Protocol**

This appendix contains instructions for configuring Service Activator to use Secure Socket Layer (SSL) protocol for HTTPS or for sending and receiving secure messages between the Workflow Manager and a Customer Relationship Management (CRM) system.

## Using SSL with Service Activator: An Overview

You can use SSL with two Service Activator components. The first is the Operator UI, which you can configure to use HTTPS. The second is the Workflow Manager, which you can configure to use SSL to send (and receive) secure messages to (and from) a CRM. The configuration processes for both of these components are similar.

### Preparing to Use SSL

Implementing a security solution such as SSL is, by nature, a complex process that involves numerous design decisions and trade-offs. This appendix does not attempt to provide a comprehensive discussion of SSL or to offer advice about how best to implement an SSL solution with Service Activator in your environment. It, instead, offers one approach that you can use to configure Service Activator to use SSL.

Before proceeding, you should be knowledgeable about SSL—in particular, using SSL with Java—in order to determine the appropriate SSL solution to use with Service Activator for your environment. The following references can assist you in understanding and implementing an SSL solution:

- The Oracle JSSE web site at <http://java.sun.com/products/jsse>
- The Oracle `keytool` reference at <http://docs.oracle.com/javase/t/doces/technotes/tools/solaris/keytool.html>
- The OpenSSL web site at <http://www.openssl.org>

### Getting Organized

Before using SSL with Service Activator, you will need to design a mechanism for using and storing keys and certificates. To do this, you will need to answer the following questions:

- What will you name your keystore?
- Where will your keystore be located?
- What will your keystore password be?
- How and where will you store trusted certificates?
- Will you use client-side authentication?
- Which Service Activator configuration files will you need to update?

### Configuring Service Activator to Use SSL

To configure either the Operator UI or the Workflow Manager to use SSL, you will need to complete the following steps:

1. Configure Java Secure Socket Extension (JSSE).
2. Create a certificate keystore.
3. Obtain and import a signed certificate into the keystore.



4. Modify the appropriate configuration files to reflect the keystore name and password.

<b>Component Using SSL</b>	<b>Configuration Files That Require Modification To Use SSL</b>
----------------------------	---

Operator UI	<code>\$JBOSS_HOME/standalone/configuration/standalone.xml</code>
-------------	---

Workflow Manager	<code>\$ACTIVATOR_ETC/config/mwfm.xml</code>
------------------	--

5. Restart Service Activator to ensure that all changes are effective.

Each of these steps will be described in detail for both the Operator UI and the Workflow Manager. For additional information about using SSL with the JBoss/Tomcat bundle, please see *JBoss Administration and Development, Second Edition*. This document is available for purchase at the [www.jboss.org](http://www.jboss.org) web site.

## Understanding the Required Software

JSSE is a reference implementation of SSL for Java. It implements the SSL and Transport Layer Security (TLS) protocols. The JAR files for JSSE are supplied by the Java run-time environment (JRE). This package also includes data encryption and server authentication functionality.

## Configuring JSSE

In the file named `$JAVA_HOME/jre/lib/security/java.security`, add the following entry if it does not already exist:

```
security.provider.#=com.sun.net.ssl.internal.ssl.Provider
```

Replace the “#” with the appropriate value based on the number of configured providers. It is essential that this value be not only unique, but also sequential starting with the value “1.” If you do not comply with this requirement, you will not be able to configure SSL correctly.

## Preparing to Load the Certificate Keystore

Tomcat currently only utilizes the Java standard Java Keystore (JKS) format. The resulting “keystore” is a repository for objects such as keys and certificates. The keystore is built using the command line Java `keytool` utility. This utility is available as part of the standard Java JDK Version 6 install. It is located in the `$JAVA_HOME/bin` directory.

For additional information about the `keytool` utility, refer to documentation located at the following URL:

<http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html>

Before you prepare your keystore, consider the following items:

- Where to store the keystore file (or files)
- What name and password to give your keystore
- Whether to use client-side authentication

---

**NOTE** The server always authenticates with the client. However, client-to-server (client-side) authentication is optional. Determine whether client-side authentication is required in your environment

---

## Managing Keys and Certificates

You can use the `keytool` utility to create, store, and manage the keys and certificates you will need to use SSL with Service Activator. There are four basic steps you will need to carry out when preparing to use SSL with either the Operator UI or the Workflow Manager:

1. Generate a new key entry. A key entry consists of a public key certificate and a private key. Key entries are stored in the keystore. When a new key entry is generated, it is added to the keystore. If the keystore does not yet exist, it is created.
2. Generate a certificate request. This request is formatted to be submitted to a Certificate Authority (CA), such as VeriSign or Thawte.
3. Send the certificate request file to a Certificate Authority (CA), such as VeriSign or Thawte, for signing.
4. Import the signed certificate into your keystore.

The following section provides a generic example of how to complete these steps using the `keytool` utility. Specific instructions for Service Activator are provided beginning on page 76.

---

**NOTE** Be sure to read the generic example carefully, as it contains important details about using `keytool` that you need to understand before you generate the keys and certificates necessary for SSL to work with Service Activator.

---

### Using the `keytool` Utility

1. Create a new key entry in the keystore named `my.keystore` with the password `mypass` using the following command:

```
keytool -genkey -keyalg RSA -alias <yourAliasName> -storepass \  
mypass -keystore my.keystore
```

The `-alias` option specifies a shortened, keystore-specific name for an entity that has a key or certificate in the keystore. The `-keyalg` option specifies the algorithm that will be used to generate the key entry; use RSA with SSL.

You will be prompted to fill in additional information including your name, organizational unit, organization, city or locality, state or province, and country. This information is used to create the distinguished name (DN) for your certificate. You will then be prompted for a key password. You can specify a password that is unique to your new key entry, or you can use the keystore password as your key password.

---

**NOTE** Only Step 1 is required to minimally configure a key and its associated certificate. Step 1 produces a self-signed certificate, which is less secure than a certificate signed by a CA. Steps 2 through 4 will replace the self-signed certificate with a certificate signed by a CA. In production environments, you are strongly encouraged to use certificates signed by a CA.

---

2. Generate a certificate request. In this case, the certificate request will be stored in the file named `my.csr`. You may specify any file name.

```
keytool -certreq -alias <yourAliasName> -file my.csr -keystore \  
my.keystore
```

You will be prompted for both the keystore password and the key password. Once you supply these passwords, you should receive the following message:

```
Certification request stored in file  
Submit this to your CA
```

3. Send the certificate request file (in this case, `my.csr`) to a Certificate Authority (CA), such as VeriSign or Thawte, for signing. Some CAs allow you to paste the contents of this file into an HTML form.
4. The CA will e-mail you a signed certificate. Save the certificate in a file. Import this file (in this case, `mysigned.cer`) into your keystore:

```
keytool -import -alias <yourAliasName> -file mysigned.cer -keystore \  
my.keystore -trustcacerts
```

This import operation replaces the self-signed certificate associated with the alias `<yourAliasName>` with the signed certificate.

---

#### NOTE

If you use a nonstandard CA, you will need to import a CA root certificate as a trusted root certificate prior to importing your own certificates into the keystore. The Java SDK ships with the file `cacerts`, which contains the most common CA root certificates. The `-trustcacerts` option allows `keytool` to use those CA certificates. To import a CA certificate into the keystore, use the following command, where `ca.crt` is the file containing the root certificate for your CA:

```
keytool -import -alias ca -file ca.crt -keystore my.keystore
```

---

## Configuring SSL for HTTPS (Operator UI)

There are three basic steps required to configure the Service Activator Operator UI to use SSL for HTTPS:

1. Load the server keystore.
2. Modify the JBoss `standalone.xml` file.
3. Start Service Activator.

Each of these steps will be described in detail in this section.

### Step 1: Loading the Server Keystore (Operator UI)

This step includes creating the keystore, obtaining a signed certificate, and importing the signed certificate into the keystore.

- a. Create a key entry in the keystore file named `activatorSSL.keystore` in the JBoss server configuration directory, `$JBOSS_HOME/server/default/conf`:

```
$JAVA_HOME/bin/keytool -genkey -alias uialias -keyalg RSA \  
-keystore $JBOSS_HOME/server/default/conf/activatorSSL.keystore
```

The suggested alias, keystore name, and keystore location shown here are not mandatory. You may use any alias, name, and location you like. The keystore location and password, however, must match those values stored in the JBoss `standalone.xml` configuration file. See “Step 2: Modifying the JBoss Configuration Files” on page 76 for additional information.

- b. Generate a certificate request, and store it in a file (in this case, `UIcert.csr`):

```
keytool -certreq -alias uialias -file UIcert.csr -keystore \  
$JBOSS_HOME/server/default/conf/activatorSSL.keystore
```

- c. Submit your certificate request to a Certificate Authority, such as VeriSign or Thawte.

- d. Upon receiving your signed certificate, save it in a file (in this case `UIsigned.cer`), and import it into your keystore:

```
keytool -import -alias uialias -file UIsigned.cer -keystore \  
$JBOSS_HOME/server/default/conf/activatorSSL.keystore -trustcacerts
```

Remember to use the same passwords in the `-import` operation that you used when you generated the key entry.

---

#### CAUTION

Be sure to check a certificate very carefully before importing it as a trusted certificate.

### Step 2: Modifying the JBoss Configuration Files

Once you have configured JSSE and loaded your certificates, you must configure JBoss to take advantage of the SSL functionality. To do this, modify the JBoss `standalone.xml` file to add an HTTPS connector. This file is located in the following directory:

```
$JBOSS_HOME/standalone/configuration
```

Add the following HTTPS Connector to the subsystem urn:jboss:domain:web:1.0:

```
<subsystem xmlns="urn:jboss:domain:web:1.0"
default-virtual-server="default-host">

  <connector name="http" protocol="HTTP/1.1" socket-binding="http"
scheme="http"/>

  <connector name="https" protocol="HTTP/1.1" socket-binding="https"
scheme="https" secure="true">

    <ssl name="keyalias" password="verySecret"
certificate-key-file="/opt/HP/jboss/my.keystore" protocol="ANY"
verify-client="false"/>

  </connector>

  <virtual-server name="default-host" enable-welcome-root="true">

    <alias name="localhost"/>

    <alias name="example.com"/>

  </virtual-server>

</subsystem>
```

Set `certificate-key-file` to the location and name you selected for your keystore, and set `password` to match your keystore password. If you want to use client-side authentication, set `verify-client` to “true.”

### Configuring the JBoss Operator UI Port

In the JBoss `standalone.xml` file, the `port` attribute is defined. By default, the `port` attribute for HTTPS is 8443. This attribute is the TCP/IP port number on which JBoss will listen for secure connections. You can change this to any port number you wish (such as the default port for HTTPS communications, which is 443).

### Step 3: Starting Service Activator

You will need to restart Service Activator to have your configuration changes take effect. To do this, follow the instructions in “Starting and Stopping Service Activator” on page 44 of the *HP OpenView Service Activator—Installation Guide*.

## Configuring SSL for Secure Message Transmission (Workflow Manager)

There are three basic steps required to configure the Service Activator Workflow Manager to send and receive secure messages using SSL:

1. Load the server keystore.
2. Modify the Workflow Manager configuration file.
3. Restart the Workflow Manager.

Each of these steps will be described in detail in this section.

### Step 1: Loading the Server Keystore (Workflow Manager)

This step includes creating the keystore, obtaining a signed certificate, and importing the signed certificate into the keystore.

- a. Create a key entry in the keystore file named `mwfmSSL.keystore` in the `$ACTIVATOR_ETCconfig` directory:

```
$JAVA_HOMEbinkeytool -genkey -alias mwfmalias -keyalg RSA \  
-keystore $ACTIVATOR_ETCconfigmwfmSSL.keystore
```

The suggested alias, keystore name, and keystore location shown here are not mandatory. You may use any alias, name, and location you like.

- b. Generate a certificate request, and store it in a file (in this case, `mfwmcert.csr`):

```
keytool -certreq -alias mwfmalias -file mfwmcert.csr -keystore \  
$ACTIVATOR_ETCconfigmwfmSSL.keystore
```

- c. Submit your certificate request to a Certificate Authority, such as VeriSign or Thawte.

- d. Upon receiving your signed certificate, save it in a file (in this case, `mfwmsigned.cer`), and import it into your keystore:

```
keytool -import -alias mwfmalias -file mfwmsigned.cer \  
-keystore $ACTIVATOR_ETCconfigmwfmSSL.keystore -trustcacerts
```

### Step 2: Modifying the Workflow Manager Configuration File

Change the values of the `keystore` and `keystore_password` parameters in the `SocketListenerModule` and `SocketSenderModule` specifications in the `mwfm.xml` file to match the keystore name and password, respectively, that you select. Also change the value of the `clientauth` parameter for the `SocketListenerModule` to reflect the type of authentication you will use. See Chapter 5, “Configuring the Workflow Manager,” on page 351 of *HP OpenView Service Activator—Workflows and the Micro-Workflow Manager* for additional information about editing this file.

### **Step 3: Restarting the HP Service Activator**

You will need to stop and restart the Workflow Manager to have your configuration changes take effect. To do this, follow the instructions in the *HP OpenView Service Activator—Installation Guide*.

## Troubleshooting

Many things can go wrong when working with JSSE and certificates. Here is a list of common problems and their solutions:

**java.security.NoSuchAlgorithmException: Algorithm SunX509 not available**  
or  
**java.security.NoSuchAlgorithmException: Algorithm TLS not available**

This common error indicates that you did not specify your security algorithm providers properly. If you configured the algorithms by modifying the `java.security` file, check to be sure that you modified the correct file and that you are executing the correct `java.exe`. Run Java with the `-version` flag to check the version number of the Java SDK you are currently using.

If the version of your SDK is correct, check the `java.security` file carefully to be sure that your `security.provider.#` line is not being overridden by another `security.provider.#` line later in the file. Next, be sure that the order of `security.provider.#` lines is sequential from 1 to #. The security manager will not recognize any provider settings if there is a gap in the number sequence.

**javax.net.ssl.SSLException: untrusted server cert chain**  
or  
**javax.net.ssl.SSLException: Received fatal alert: certificate\_unknown**

These exceptions will be thrown if a server or client is unable to validate the credentials provided by the other party. For instance, if a certificate is not signed by any other certificates known (and trusted) by the trust manager, the certificate will be rejected. If you are having this problem with two parties that should be trusting each other, verify that each certificate has been imported into the keystore of the other and that the certificate authority used to sign each certificate has been distributed properly.

**java.io.IOException: Keystore was tampered with, or password was incorrect**

This error typically indicates that the password provided to retrieve the certificates from the local keystore is incorrect, but it could also mean that something is wrong with the keystore file itself. The file might be corrupted, or the file permissions might be too restrictive.

**javax.net.ssl.SSLException: No available certificate corresponds to the SSL cipher suites which are enabled.**

This exception is typically thrown when a connection is being initialized. It means that a socket or server socket object does not have any certificates, or not the right kind of certificates, to use when starting communication or listening on the port. To solve this, make sure that the keystore file is being loaded correctly, that it is the keystore you intended to use, and that the context is initialized with the right set of key and trust managers.



### Client Hangs While Connecting

The client may hang if it is trying to use a cleartext socket, but the server is using TLS. Since the server is expecting a stream containing protocol negotiation data, it will wait on the open socket until it hears what it is listening for. Eventually the client will time out.

### Finding Additional Information

If you experience a problem with your SSL implementation that is not addressed by one of the solutions discussed in this section, examine the following log files for further information:

Component	Log Files To Examine
Operator UI or JBoss	boot.log server.log.*
Workflow Manager	mwfm_active.log.xml



---

## **B** **mwfmtool**

In this appendix, you can find all the necessary information about `mwfmtool`. The chapter contains the complete list of the commands together with the parameters they accept.

---

## mwfmtool

mwfmtool is used to issue commands and receive results from Service Activator. Using the tool, you can achieve exactly the same results as working in the Operator UI, i.e. you can control the Workflow Manager in exactly the same way as you would do in the Operator UI of Service Activator.

mwfmtool would normally be used by system administrators for testing purposes, or other administrative tasks. You can also use mwfmtool for integration with other applications, e.g. to start workflows in Service Activator from other external applications.

mwfmtool is a command line tool. It does not have any graphical user interface. There are no particular tasks for which you must specially use mwfmtool. It is your personal preferences that determine how (via mwfmtool or Operator UI) you communicate with Service Activator. Note, however, that mwfmtool allows you to run scripts, which automates your routine tasks but requires some scripting or programming experience. Please see Example B-2 on page 511.

---

### NOTE

The tool is called `mwfmtool.bat` in Windows. UNIX users must run `mwfmtool`.

### Start mwfmtool

These are the steps to start mwfmtool.

1. Start your command line interface.
2. Change directory to `$ACTIVATOR_BIN`
3. Enter `mwfmtool` and press **Return**.
4. The help line appears, which details the command syntax.

```
<cmd> [-host<hostname>] [-port<port>]  
[-user<user>[/<password>]] [-quiet]<...cmd args...>
```

You can find more information about the command syntax in Table B-2 on page 503.

If you want to see the list of the commands available in mwfmtool, enter `mwfmtool a`. In this case, “a” is a simple character selected at random. It provokes an error (“a” is not a valid command in mwfmtool), to which mwfmtool responds by displaying the complete list of the valid commands.

You can also get help on using individual commands. For example, to see what parameters the `KillJob` command takes, enter `mwfmtool KillJob` and enter **Return**. Normally, the command takes several parameters. In this case, you do not provide any of them. This provokes an error, to which mwfmtool responds by displaying help. The result of entering `KillJob` without parameters looks like so:

```
error: missing expected parameter  
usage: KillJob [-user<username>[/password]]  
[-host<hostname>] [-port<port>] [-quiet] <jobID>
```

---

### NOTE

mwfmtool processes a single command at a time. When you enter commands, remember to begin your command line with `mwfmtool` followed by the command name.

## Using mwfmttool from a Remote Computer

If you use `mwfmttool` on the computer where Service Activator is installed, then you can skip this section.

You can, however, use `mwfmttool` without installing Service Activator. In this case, you would use `mwfmttool` to connect to a remote computer, on which Service Activator is installed, and issue commands to that computer.

To be able to do this, you will need to copy the four files listed in Table B-1 on page 501 to your computer without Service Activator. The column “Location” in Table B-1 contains the locations where the files can be found in a typical installation of Service Activator. For further instructions, see “Move mwfmttool to a Computer Running UNIX” on page 501 or “Move mwfmttool to a Computer Running Microsoft Windows” on page 502.

**Table B-1**

**mwfmttool Files**

File	Location
<code>mwfmttoolusage.txt</code>	<code>\$ACTIVATOR_ETC/config</code>
<code>mwfmRB_en.properties</code>	<code>\$ACTIVATOR_ETC/nls</code>
<code>activator_utils.jar</code>	<code>\$ACTIVATOR_OPT/lib</code>
<code>mwfm.jar</code>	<code>\$ACTIVATOR_OPT/lib</code>

## Move mwfmttool to a Computer Running UNIX

This section has the instructions on how you can move `mwfmttool` to a computer running UNIX. In these instructions, the computer which has Service Activator installed is referred to as Computer A; the computer to which `mwfmttool` is moved is referred to as Computer B.

1. Install a supported Java version on Computer B and set the environment variable `JAVA_HOME`.
2. Create the following directories in Computer B
  - `/opt/mwfmttool/bin`
  - `/opt/mwfmttool/etc/config`
  - `/opt/mwfmttool/etc/nls`
  - `/opt/mwfmttool/lib`
3. Locate the files listed in Figure B-1 in Computer A.
4. Copy the files `mwfm.jar` and `activator_utils.jar` to the directory `/opt/mwfmttool/lib`
5. Copy the `mwfmttoolusage.txt` to the directory `/opt/mwfmttool/etc/config`
6. Copy the `mwfmRB_en.properties` to the directory `/opt/mwfmttool/etc/nls`
7. Create a script called `mwfmttool` in the directory `/opt/mwfmttool/bin`. The content of the script must be the following

```
#!/bin/bash

ACTIVATOR_ETC=/opt/mwfmttool/etc
```

```
CLASSPATH=/opt/mwfmtool/lib/mwfm.jar  
CLASSPATH=$CLASSPATH:/opt/mwfmtool/lib/activator_utils.jar  
CLASSPATH=$CLASSPATH:$ACTIVATOR_ETC/nls  
$JAVA_HOME/bin/java -classpath $CLASSPATH \  
-DMWFMTTOOL_ETC=$ACTIVATOR_ETC \  
com.hp.ov.activator.mwfm.client.mwfmtool "$@"
```

8. To start `mwfmtool`, follow the instructions in “Start `mwfmtool`” on page 500.

### Move `mwfmtool` to a Computer Running Microsoft Windows

Below are the instructions on moving `mwfmtool` to a computer running Microsoft Windows. In these instructions, the computer which has Service Activator installed is referred to as Computer A; the computer to which `mwfmtool` is moved is referred to as Computer B.

1. Install a supported Java version on Computer B and set the environment variable `JAVA_HOME`.

2. Create the following directories in Computer B

- C:\HP\OpenView\mwfmtool\bin
- C:\HP\OpenView\mwfmtool\etc\config
- C:\HP\OpenView\mwfmtool\etc\nls
- C:\HP\OpenView\mwfmtool\lib

3. Locate the files listed in Figure B-1 in Computer A.

4. Copy the files `mwfm.jar` and `activator_utils.jar` to the directory  
C:\HP\OpenView\mwfmtool\lib

5. Copy the `mwfmtoolusage.txt` to the directory  
C:\HP\OpenView\mwfmtool\etc\config

6. Copy the file `mwfmRB_en.properties` to the directory  
C:\HP\OpenView\mwfmtool\etc\nls

7. Create a script called `mwfmtool.bat` in the directory  
C:\HP\OpenView\mwfmtool\bin. The file content must be as follows.

```
@echo off  
  
set ACTIVATOR_ETC=C:\HP\OpenView\mwfmtool\etc  
  
set CLASSPATH=%ACTIVATOR_ETC%\nls  
  
set CLASSPATH=%CLASSPATH%;C:\HP\OpenView\mwfmtool\lib\mwfm.jar  
  
set CLASSPATH=%CLASSPATH%;C:\HP\OpenView\mwfmtool\lib\activator_utils.jar  
  
%JAVA_HOME%\bin\java -classpath %CLASSPATH% \  
-DMWFMTTOOL_ETC=%ACTIVATOR_ETC% com.hp.ov.activator.mwfm.client.mwfmtool %*
```

8. To start `mwfmtool`, follow the instructions in “Start `mwfmtool`” on page 500.

## mwfmtool Commands

Here you can find the information about the structure of the commands in `mwfmtool` as well as the complete list of these commands.

`mwfmtool` is not case sensitive. You can enter command names in upper or lower case. Some commands have abbreviations, which you can use instead of the full name of a command. Some of the commands may also have several abbreviations.

Commands have arguments, which control how commands are executed and what output they return. If a command has several arguments, then those arguments are separated by spaces only. No other punctuation marks between arguments are used.

Some of the command arguments are optional, i.e. you do not have to supply them. In this document, optional arguments are enclosed in brackets, e.g. `[-host<hostname>]`. The obligatory values are enclosed in less than (<) and greater than (>) symbols, e.g. `<hostname>` in `[-hostname<hostname>]` indicates that you must always provide a host name when using the `-hostname` argument. Remember though, these symbols (`[ ] < >`) are used for the purposes of this document only. They are not used when entering commands, e.g. a valid command is `mwfmtool GetJobStatus 123`.

Finally, the pipe character (`|`) separating two arguments or values indicates that you must use one of those two arguments or values when entering a command.

**Table B-2** Command Structure

Parameter	Description
<code>&lt;cmd&gt;</code>	Command name. See Table B-3 on page 504, which contains the command names.
<code>[-host&lt;hostname&gt;]</code>	Host name. By default, <code>mwfmtool</code> assumes that you try to log on to the local host. If you want to connect to another computer, provide the host name here, e.g. <code>-host *.*.*.*</code>
<code>[-port&lt;port&gt;]</code>	Port name. If left out, <code>mwfmtool</code> assumes the default port of the Workflow Manager. In typical installations, it is 2000. If your Service Activator is set up otherwise, provide here the port number used by the Workflow Manager, e.g. <code>-port 2001</code>
<code>[-user&lt;username&gt;[/&lt;password&gt;]]</code>	The user name and the password you use to log on to Service Activator when user name and password authentication is enabled, e.g. <code>-user aaa/**</code> In this example, “aaa” is the user name while the asterisks represent the password.
<code>[-quiet]</code>	This parameter allows you to control the output returned by the commands. It accepts either <code>-verbose</code> or <code>-quiet</code> as its values. Use <code>-verbose</code> if you want your commands to generate detailed information about their results. Use <code>-quiet</code> to turn off all output.

**Table B-2 Command Structure (Continued)**

Parameter	Description
<...cmd args...>	The mandatory value the command takes. For example, when you use command DeleteMessage, you must provide the message ID and the name of the queue in which the message is. You can find the commands with their arguments in Table B-3 on page 504.

**Table B-3 Command List**

Command	Abbreviation	Description
canKillJob <jobID>	not available	Check whether the user has the correct role to kill the job
ChangeJobRoles <jobID> <defaultRole> [<traceRole>] [<killRole>]	not available	Changes roles of a running job.
ChangeRequestRole <messageId> <queue> <role>	changereqrole	Changes the role of a current request. The role is changed to the one you supply in the <role> argument.
DeleteAllMessages [-queue <queue>] [-priorto <date>   <seconds>]	delallmessages delallmsgs delallmsg	Deletes all messages in a given queue or all queues. The -priorto argument indicates that messages posted earlier than a given time must be deleted. You can specify a date or a number of seconds prior to now. CAREFUL: If you enter the command without any arguments, all messages are deleted immediately! mwfmtool does not additionally warn that you are about to delete messages.
DeleteMessage <messageID> <queue>	delmsg	Deletes a message from a queue. You must provide the message ID and the queue name.
GetAllClusterNodes	not available	Get the host names of all the cluster nodes
GetCasePacketForJob <jobID>	not available	Get all the case packets for the given job



**Table B-3 Command List (Continued)**

Command	Abbreviation	Description
GetCounters	not available	Get current running job count, scheduled job count, total jobs waiting for activation and total jobs waiting for user input (all cluster nodes inclusive)
GetCurrentJobCount	not available	Get the current count of running jobs (all cluster nodes inclusive)
GetFullThreadsDump	not available	Get a full thread dump from a cluster node.
GetHistoricalSystemData	not available	Get the historical system data of one of the cluster nodes: memory (heap and non heap), worker threads, activation threads, activation queue, total jobs, user sessions and database pools.
GetJobStatistics <jobID>	not available	Get the statistics about all the jobs running in all the cluster nodes
GetJobStatus <jobID>	not available	Gets the status of a job.
GetJobDefaultRole <jobID>	not available	Gets the default role for a given job.
GetJobTraceRole <jobID>	not available	Gets the trace role for a given job.
GetJobKillRole <jobID>	not available	Gets the kill role for a given job.
GetJobRequestRole <jobID> <queue>	not available	Gets the role of a request waiting on a queue.
GetMaster	not available	Get the cluster node which is acting as a master for the given module name. The module name will have to match the modules defined in mwfm.xml
GetLogFileHostInfo -i <unique-ID>	not available	Get the name of the log file name / hostname in which the log entry with <unique-ID> is present
GetNextProcessId	not available	Get the next job Id from the database

**Table B-3 Command List (Continued)**

Command	Abbreviation	Description
GetQueueCount <queue name> <ignore role>	not available	Get the number of jobs in a given queue across all cluster nodes (except Running Jobs and Scheduled Jobs). If role has to be ignored, set ignore role to be true. If role is needed, set it to false.
GetRunningJobCount <local cluster node / all cluster nodes>	not available	Get the number of running jobs for the given user (across all cluster nodes). True for local cluster node; false for all cluster nodes
GetScheduledJobCount	not available	Get the number of scheduled jobs (across all cluster nodes)
GetStatusForAllClusterNodes	getstatus	Get the ONLINE, LOCKED and SUSPENDED state of all the cluster nodes
GetSystemData	not available	Get the current system data of one of the cluster nodes: memory (heap and non heap), worker threads, activation threads, activation queue, total jobs, user sessions and database pools.
GetValidRoles	not available	Gets the list of valid roles according to the Authentication module.
GetVersion	not available	Get the version of the Service Activator application
GetWorkflowInfo <workflow name>	not available	Get the information about a given workflow.
KillJob <jobID>	not available	Kills a given job.
ForceKillJob <jobId>	not available	Forces a job to stop even if it is blocked. Use this command ONLY if KillJob fails to stop the job.
ChangePriority -jobId <jobId> -priority <priority>	not available	Changes the priority of a given job
ListMessageQueues	listmsgqueues	Shows the list of all message queues.
ListRequestQueues	not available	Shows the list of all request queues.

**Table B-3 Command List (Continued)**

Command	Abbreviation	Description
LoadWorkflows	not available	Loads all the workflows in all the cluster nodes
Lock <host name>	not available	Lock the given host name.
LockAllNodes	not available	Locks all the nodes in the cluster
ReloadConfiguration	not available	Will reload configuration for both mwfm and resmgr in all the cluster nodes
ResumeAllNodes	not available	Will initiate a state change from suspend to resume of all the nodes in the cluster
ResumeNode <host name>	not available	Will initiate a state change from suspend to resume in the given host
SuspendAllNodes	not available	Will initiate a state change from resume to suspend of all the nodes in the cluster
SuspendNode <host name>	not available	Will initiate a state change from resume to suspend in the given host
UnlockAllNodes	not available	Unlocks all the nodes in the cluster
UnlockNode <host name>	not available	Unlock the given host name.
SendCasePacket <jobID> <queue>	not available	Passes values to a job waiting in a given request queue. This command has a dialog for entering the variables. You can enter and pass multiple variables. See Example B-1 on page 511 for more details.
ShowDatabaseMessages [-messageId <messageId>] [-identifier <identifier>] [-jobId <jobId>] [-hostName <hostName>] [-moduleName <moduleName>]	showdbmsgs	Shows the entire row information in the database_message table for given input parameters like identifier, jobId, hostName, moduleName. Identifier is provided by the solution and is used to map the database record to the specific request if no parameter is specified for this command, this API will fail!

**Table B-3 Command List (Continued)**

Command	Abbreviation	Description
DeleteDatabaseMessages [-messageId <messageId>] [-identifier <identifier>] [-jobId <jobId>] [-hostName <hostName>] [-moduleName <moduleName>]	deletedbmsgs	Deletes entries in the database_message table for given input parameters like identifier, jobId, hostName, moduleName, messageId. Identifier is provided by the solution and is used to map the database record to the specific request if no parameter is specified, this API will fail!
GetAllNodesTimeStatus -status <overall   report   complete>		overall : will give overall time sync status (All the nodes are in sync if the system time of all the cluster nodes are same (or) Time mismatch between the nodes if there is a time difference between the node report : will give time sync status report of all node. The report will have each node name and its time sync status
ShowJobDescriptor -job<jobID> -queue<name> [casePacketVars<var1><var2>... <varN>]	not available	Shows the job descriptor for a given job in a queue. In addition to the general output, you can request the value of a certain set of case-packet variables.
ShowMessages [-verbose] [-queue<queue>]	showmsg showmsgs	Shows all messages posted in a given queue or all queues.
ShowRequests [-verbose] [-queue<queue>] [casePacketVars<var1><var2> ...<varN>]	not available	Shows all requests waiting in a given queue (or all queues). In addition to the general output, you can request the value of a certain set of case-packet variables.
ShowRunningJobs [-verbose] [-job<jobID>] [-casePacketVars<var1><var2> ...<varN>]	showjobs	Shows details about the state of a given job. In addition to the general output, you can request the value of a certain set of case-packet variables.
QueryMessages [-serviceId <serviceId>] [-orderId <orderId>] [-type <type>] [-state <state>] [-maxRecords <max records>]	showmsg showmsgs	Shows the messages which match the query parameters

**Table B-3 Command List (Continued)**

Command	Abbreviation	Description
QueryRequestQueueJobs -queue <queueName> [-serviceId <serviceId>] [-orderId <orderId>] [-type <type>] [-state <state>] [-maxRecords <max records>]	showjobs	Shows the jobs in a queue which match the query parameters
QueryRunningJobs [-serviceId <serviceId>] [-orderId <orderId>] [-type <type>] [-state <state>] [-maxRecords <maxRecords>]	showjobs	Shows the running jobs which match the query parameters
QueryScheduledJobs [-serviceId <serviceId>] [-orderId <orderId>] [-type <type>] [-state <state>] [-maxRecords <maxRecords>]	not available	Shows the scheduled jobs which match the query parameters
ShowWorkflows [-verbose] [-reload]	showwf	Shows all workflows defined in the system. The user can start any of them. The -reload argument tells the system to reload the workflows.
StartJob [-wait] [-noinput   -messageFile<file>] [-repeat] <workflowName>	not available	Starts a workflow. The -wait argument tells the command to wait until the job completes. The -noinput argument tells the command not to prompt for initial case-packet values. The -messageFile argument sets the message_file case-packet variable to a given file. If -noinput is NOT set, then the user is prompted for a list of the initial arguments. The -repeat argument indicates to the command that the job must be repeated N times. This argument is primarily used for testing purposes.

**Table B-3 Command List (Continued)**

Command	Abbreviation	Description
<pre>ScheduleJob -time&lt;timestamp&gt; [repeatingPeriod&lt;period&gt; -repeatingPeriodUnit &lt;unit&gt; -repeatingEnd&lt;end&gt; -repeatingType &lt;type&gt;] [-description&lt;description&gt;] [-groupID&lt;groupID&gt;] [-status&lt;status&gt;] [[-start_missed_scheduled_instances &lt;true false&gt;] [-noinput   -messageFile&lt;file&gt;] &lt;workflowName&gt;</pre>	not available	<p>Schedules a workflow. The -time argument tells the command at which time to schedule a job.</p> <p>-repeatingPeriod is a period of time in milliseconds after which the job must be started again. -repeatingEnd is the timestamp in milliseconds at which repeating ends.</p> <p>-repeatingType indicates whether the Scheduled Time calculated for reoccurring jobs on restarts is relative or absolute. Allowed types: 1-relative, 2-absolute</p> <p>-groupID is used to add several jobs to a group. -description is a short description of a job.</p> <p>-status marks the current status of a job.</p> <p>-start_missed_scheduled_instances option Controls whether missed scheduled instance must be started on restart Allowed values: true, false. -noinput tell the command not to prompt for the initial case-packet values.</p> <p>-messageFile sets the message_file case-packet variable to the given file.</p>
<pre>GetScheduledJob &lt;jobID&gt;</pre>	not available	Returns all attribute of a scheduled job.
<pre>DeleteScheduleJob &lt;jobID&gt;</pre>	not available	Deletes a scheduled job.
<pre>ModifyScheduleJob [-startTime&lt;time&gt;] [-repeatingPeriod&lt;period&gt;] [-repeatingEnd&lt;end&gt;] [-description&lt;description&gt;] [-groupId&lt;groupID&gt;] [-status&lt;status&gt;]</pre>	not available	<p>Modifies a scheduled workflow. The -startTime argument tells the command at which time to schedule a job.</p> <p>-repeatingPeriod is a period of time in milliseconds after which the job must be started again. -repeatingEnd is the timestamp in milliseconds at which repeating ends.</p> <p>-groupID is used to add several jobs to a group. -description is a short description of a job.</p> <p>-status marks the current status of a job.</p>

**Table B-3 Command List (Continued)**

Command	Abbreviation	Description
MakePrimarySite <site name>	not available	Makes the specified site as the primary site when Service Activator clusters are run in a disaster recovery setup.
SearchLog -q <search-query> -s <max-results> [-r]	not available	Search log index and write result to STDOUT. <search-query> is the query to use for searching (supports Apache Lucene syntax). <max-results> is the maximum number of search results to display. If the 'r' option is specified the search results are listed in reverse order (newest entry first)

Below you can find an example of how a command is used.

**Example B-1 SendCasePacket**

In this example, you start the `SendCasePacket` command and enter a number of variables, which you then pass to a job.

As it has been mentioned, the command requires the job ID and the name of the queue, on which the job can be found. In this example, the job ID is 111149345, the queue name is “queue1”. It is assumed that the job waits for three variables called `variable1`, `variable2`, `variable3`, which have values 1, 2 and 3 respectively. Note that you must know the exact variable names. `mwfmtool` does not check correctness of the variables and their values as they are entered.

Remember to end variable input with an empty line, i.e. once you have entered the last variable, press **Return** to get another empty line for variable input then press **Return** again to send the variables. The command dialog looks similar to this:

```
C:\HP\OpenView\ServiceActivator\bin>mwfmtool SendCasePacket 111149345 queue1

Enter values for case-packet variables to be initialized.
Expected input is of the form <variable>=<value>.
Use an empty line to finish input.

variable:  variable1=1
variable:  variable2=2
variable:  variable3=3
variable:

Sending 3 parameters to job 111149345
```

**Example B-2 Running a Script**

Assume that many jobs are waiting for interaction in the “ScriptQueue” queue. They all wait for the input parameter `Request`. This example shows how you can set a value for all the jobs by using `mwfmtool`.

```
@echo off
    call mwfmtool ShowRequests -queue "ScriptQueue" -cpv JOB_ID > temp.txt
    FOR /F "tokens=2" %%a IN (temp.txt) DO call :stopp %%a
    del temp.txt cvp.txt

goto :done

:stop
    @echo Request=stop > cvp.txt
    call mwfmtool SendCasePacket %1 "ScriptQueue" < cvp.txt > NULL
    echo hob %1 stopped
    goto : done

:done
```



---

## **C** **Creating Additional Data Source**

If an extra data store has to be added then a new element must be added to the `$JBOSS_HOME/standalone/configuration/standalone.xml`.

In this file find all the predefined datasources delivered with HPSA can be found. They can be found in the Datasources sub-system. Search e.g. for the jndi-name attribute with the name `java:/hpsa/jdbc/mwfmDB`. Then make a copy of this element and modify the jndi-name, pool-name, and security-domain. The jndi-name must start with `java:/hpsa/jdbc` to be possible to use as a new inventory datasource.

Apart from adding the datasource element to the standalone.xml file a new security-domain element must be added in the sub-system security-domains. Take a copy of one of the predefined HPSA security elements and modify the name to reflect what was configured in the new datasource element. The finally the username and password parameters must be modified.

Use the `$ACTIVATOR_BIN/generateEncrypted[.bat]` utility to create an encrypted password.

```
#!/generateEncrypted.sh -password ovsapassword
# Encoded password: 340eafbedf6d293cc3bc376bef610c0a
```

Now the new datasource can be used to specify an additional database module in the `$ACTIVATOR_ETC/config/mwfm.xml`.

```
<Module>
<Name>newdbmodule</Name>
<Class-Name>com.hp.ov.activator.mwfm.engine.module.JNDIDatabaseModule
</Class-Name>
<Param name="datasource_jndi_name" value="my-jndi-name" />
</Module>
```

**A**

- activation
  - description, 357
- adding values together in a workflow, 100
- AlarmModule Methods, 478
- AskFor, workflow node, 104
- Assign, workflow node, 109, 113, 134, 216
- assigning database results to a case-packet variable, 145
- assigning values to case-packet variables, 109, 216
- Audit, 28
  - audit module, 365
  - automatically generated, 28
  - collecting records, 64
  - node, 110
  - node parameters, 110
  - records, 28
- authentication
  - description, 357
  - HP-UX module, 378, 399, 444
- authentication and authorization
  - writing your own modules, 473
- authentication methods, 473
- authorization
  - HP-UX module, 378, 399, 444

**C**

- case-packet variables
  - assigning values, 109, 216
  - database values, assigning, 145
  - description, 35
  - mapping fields from an XML message, 329
  - passing to a running program, 148
  - reading text files into, 274
  - setting based on templates, 323
  - storing in a database table, 210
  - types of, 35
- characters, substituting, 280
- collecting statistics, 407
- ComposeMessage, workflow node, 110, 114
- ConfirmResourceReservation, workflow node, 118, 121
- contacting the activation engine from a workflow, 96
- conventions
  - typographical, 15
- creating a micro-workflow manager client, 483

**D**

- database access, description, 358
- Default role, 29
- delete instance parameters from the repository, 137
- DeleteServiceInstance, workflow node, 102, 123, 137
- deleting a file, 277, 279
- DoNothing, workflow node, 119, 140

**E**

- e-mail messaging, 403, 405
- ending a workflow, 201
- enhancing existing functionality, 466
- Equal, rule node, 144
- ExecSQLQuery, workflow node, 145
- ExecSQLStatement, workflow node, 147
- ExecuteExternal, workflow node, 148, 150
- external communication
  - configuring, 428
  - opening a socket, 428
- external interfaces
  - listener module, 429, 432
  - sending a message, 432

**F**

- file, removing, 277, 279

**G**

- GreaterThan, workflow node, 174
- GreaterThanOrEqual, workflow node, 175, 176, 179

**H**

- handlers, 33
  - library of, 339
  - ReleaseResourceHandler, 345
  - SyncHandler, 347
- HpuxAuthModule, 369, 371, 374, 377, 378, 394, 399, 436, 444

**I**

- incoming messages
  - opening a socket, 428
- incrementing values in workflows, 100
- interacting with micro-workflow manager, 483

**K**

- Kill role, description, 29
- KillJob, workflow node, 201

**L**

- LessThan, workflow node, 202
- LessThanOrEqual, workflow node, 203, 204
- library of micro-workflow manager modules, 362
- linking the micro-workflow manager with activation, 363
- logging
  - XMLLogModule, 416, 421, 426, 434, 448

**M**

- mapping XML message fields to case-packet variables, 329
- MatchDBQuery, workflow node, 206, 208
- MatchDBStore, workflow node, 210
- messages

- composing with case-packet variables, 114
  - request and message, 405
  - sending, 291
  - sending to OVO, 415
  - micro-workflow manager
    - creating a client, 483
    - interacting with, 483
    - modules, description, 22
    - writing your own modules, 466
  - micro-workflow manager modules
    - activation, 357
    - authentication, 357
    - database access, 358
    - library, 362
    - transaction state, 357
    - writing your own, 466
  - modules
    - activation, description, 357
    - authentication, description, 357
    - database access, 358
    - DatabaseModule, 389
    - HpuxAuthModule, 369, 371, 374, 377, 378, 394, 399, 436, 444
    - micro-workflow manager, library, 362
    - Monitor, 407
    - OVOMessageModule, 415
    - queue hook, writing your own, 476
    - sender, writing your own, 477
    - transaction state, 357
    - XMLLogModule, 416, 421, 426, 434, 448
  - Monitor, 407
  - MoveFile, workflow node, 215
  - moving or renaming a file, 215
  - Multiply, workflow node, 217, 218, 219, 221, 222
- N**
- nodes, workflow, 20
  - Not, workflow node, 223, 225, 226, 227, 228, 229, 245, 246, 247
  - notification by e-mail, 405
- O**
- opening a socket for incoming messages, 428
  - OpenView Operations
    - message module, 415
  - Operator UI
    - showing statistics, 380
    - viewing statistics, 407
  - OVOMessageModule, 415
- P**
- parameters for micro-workflow manager, 22
  - pausing a workflow, 294
  - preventing more than one workflow instance, 27
  - process nodes, 30
    - writing custom, 454
  - PutMessage, workflow node, 249
- Q**
- querying inventory from a workflow, 253
  - querying the database, 145
  - QueryInventory, workflow node, 253
  - QueryServiceInstance, workflow nodes, 259
  - QueueHook Methods, 476
- R**
- ReadFile, workflow node, 274
  - reading text files into a case-packet, 274
  - relational database module, 389
  - ReleaseResource, workflow node, 276
  - ReleaseResourceHandler, workflow node, 345
  - releasing poolable resources, 276
  - releasing resources, 118, 121, 345
  - RemoveFile, workflow node, 277, 279
  - removing resources from the
    - RESERVATIONS variable, 118, 121
  - renaming or moving a file, 215
  - Replace, workflow node, 280
  - reservable resources
    - releasing, 276, 345
    - reserving, 281
  - ReserveResource, workflow node, 281
  - reserving resources, 281
  - restricting workflows to one instance at a time, 27
  - retrieving values of service-instance parameters, 259
  - roles
    - description, 29
  - rule nodes
    - , 30
    - Equal, 144
    - GreaterThan, 174
    - GreaterThanOrEqual, 175, 176, 179
    - LessThan, 202
    - LessThanOrEqual, 203, 204
    - Not, 223, 225, 226, 227, 228, 229, 245, 246, 247
    - writing custom, 458, 460
  - running a SQL statement against a database, 147
- S**
- SendAlarm, workflow node, 289
  - sender module
    - writing your own, 477
  - SenderModule Methods, 477
  - sending a message to a workflow module, 289
  - sending e-mail notification, 405
  - sending messages, 432
  - sending messages to OVO, 415
  - sending messages using SenderModule, 291

---

- SendMessage, workflow node, 291
- service-instance
  - retrieving values of, 259
- service-instance parameters
  - updating, 314
- service-instance repository
  - deleting instance parameters, 137
  - setting new values, 318
  - updating, 318
- setting case-packet variables based on templates, 323
- setting new values for technical parameters, 318
- showing statistics in the Operator UI, 380
- Sleep, workflow node, 294
- specifying a wait period for user interaction, 104
- SQL query, 145
- SQL statement, running, 147
- SSL communication, 429, 432
- Start role, 29
- starting workflows automatically, 27
- StartJob, workflow node, 295
- startup attributes for workflows, 27
- statistics
  - collecting, 407
  - viewing in the Operator UI, 407
- storing case-packet contents in a database table, 210
- substituting characters, 280
- SyncHandler, workflow node, 347
- synchronizing child and parent workflows, 347

## T

- TCP communication, 429, 432
- text files, reading into a case-packet, 274
- Trace role
  - description, 29
- transaction state, description, 357

## U

- UpdateInProgress, workflow node, 314
- UpdateServiceInstance, workflow node, 318
- updating service-instance parameter values, 314
- user interaction, pausing the workflow, 104

## V

- values
  - retrieving, 259
- values, multiplying in a workflow, 217, 219, 222
- VariableMapper, workflow node, 323

## W

- workflow
  - modules, sending messages to, 289
- workflow nodes, 20, 30

- writing custom, 465
- workflows
  - a programming analogy, 24
  - Add node, 100
  - AskFor node, 104
  - contacting the activation engine, 96
  - ending, 201
  - handler library, 339
  - multiplying values, 217, 219, 222
  - nodes, 30
  - nodes, description, 20
  - pausing, 294
  - preventing more than one instance, 27
  - querying inventory, 253
  - restricting instances, 27
  - starting automatically, 27
  - startup attributes, 27
- WriteCasePacket, workflow node, 326
- writing
  - custom process nodes, 454
  - custom rule nodes, 458, 460
  - custom workflow nodes, 465
  - micro-workflow manager modules, 466
  - sender module, 477

## X

- XMLMapper, workflow node, 329, 334

