****

HPE Service Activator

Workflows and the Workflow Manager

for Microsoft Windows Server 2019®, CentOS 7.6 (or later 7.x versions) and Red Hat Enterprise Linux 7.6 (or later 7.x versions) operating systems

Release:

January 15, 2020



# Notices

**Legal notice**

© Copyright 2001-2019 Hewlett Packard Enterprise Development LP

Confidential computer software. Valid license from HPE 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 HPE 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. HPE shall not be liable for technical or editorial errors or omissions contained herein.

Printed in the US

**Trademarks**

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

OpenJDK is a registered trademark of Oracle and/or its affiliates.  
Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.   
Oracle® and Java™ are registered trademarks of Oracle and/or its affiliates.  
EnterpriseDB® is a registered trademark of EnterpriseDB.  
EDB Postgres® Advanced Server is a registered trademark of EnterpriseDB.  
UNIX® is a registered trademark of The Open Group.  
JBoss® is a registered trademark of Red Hat, Inc. in the United States and other countries.  
WildFly® is a registered trademark of Red Hat, Inc. in the United States and other countries.  
Microsoft® is a U.S. registered trademark of Microsoft Corporation.  
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.

\*This is a trademark that is owned by Micro Focus International PLC or one of its affiliates.

# Contents

[Notices 2](#_Toc30015737)

[Contents 3](#_Toc30015738)

[List of Tables 11](#_Toc30015739)

[List of Figures 18](#_Toc30015740)

[Install Location Descriptors 20](#_Toc30015741)

[Conventions 21](#_Toc30015742)

[In This Guide 22](#_Toc30015743)

[Chapter 1 Understanding Workflows and Workflow Manager 23](#_Toc30015744)

[1.1 What is a Workflow? 23](#_Toc30015745)

[1.2 Workflow Nodes 23](#_Toc30015746)

[1.3 Understanding the Workflow Manager 24](#_Toc30015747)

[1.3.1 Workflow Manager Modules 24](#_Toc30015748)

[1.4 Programming Analogy 24](#_Toc30015749)

[Chapter 2 Creating and Deploying Workflows 26](#_Toc30015750)

[2.1 Understanding Workflows 26](#_Toc30015751)

[2.1.1 General Structure of a Workflow 26](#_Toc30015752)

[2.1.2 Startup Attributes 27](#_Toc30015753)

[2.1.3 Workflow Persistence Attribute 27](#_Toc30015754)

[2.1.4 Audit and Statistics Attributes 27](#_Toc30015755)

[2.1.5 Setting Roles 28](#_Toc30015756)

[2.1.6 Workflow Nodes 28](#_Toc30015757)

[2.1.6.1 Process Nodes 29](#_Toc30015758)

[2.1.6.2 Rule Node 30](#_Toc30015759)

[2.1.6.3 Switch Nodes 30](#_Toc30015760)

[2.1.7 Handlers 31](#_Toc30015761)

[2.1.8 Conventions for Node and Handler Parameters 31](#_Toc30015762)

[2.1.9 Case-Packet Variables 32](#_Toc30015763)

[2.1.10 Initial Case-Packet Values 33](#_Toc30015764)

[2.1.11 Workflow Contract 33](#_Toc30015765)

[2.1.12 Default Case-Packet Variables 34](#_Toc30015766)

[2.1.13 References to Workflows Data Types in Workflow Node Parameters 39](#_Toc30015767)

[2.1.13.1 Arrays or Lists 39](#_Toc30015768)

[2.1.13.2 Collections 39](#_Toc30015769)

[2.1.13.3 Java Beans 39](#_Toc30015770)

[2.1.13.4 Maps 39](#_Toc30015771)

[2.1.14 Queues 40](#_Toc30015772)

[2.1.14.1 Queue Names 41](#_Toc30015773)

[2.1.14.2 Queues and Roles 41](#_Toc30015774)

[2.2 Advanced Workflow Techniques 41](#_Toc30015775)

[2.2.1 Spawning Child Workflows 41](#_Toc30015776)

[2.2.2 Using Timeouts 44](#_Toc30015777)

[2.2.3 Using Priorization in Workflows 45](#_Toc30015778)

[2.2.3.1 Priorizing Workflow Node Processing 45](#_Toc30015779)

[2.2.4 Uploading Data from a Task Activation 47](#_Toc30015780)

[2.3 Deploying Workflows 47](#_Toc30015781)

[2.4 Clustering Considerations 48](#_Toc30015782)

[Chapter 3 Using the Workflow Designer 49](#_Toc30015783)

[3.1 Navigating the Workflow Designer 49](#_Toc30015784)

[3.1.1 Understanding Workflow Designer Features 50](#_Toc30015785)

[3.2 Using the Main Menu 51](#_Toc30015786)

[3.3 Using the Main Utilities Toolbox 53](#_Toc30015787)

[3.4 Using the Visual Properties Toolbox 53](#_Toc30015788)

[3.5 Using the Context Sensitive Menu 55](#_Toc30015789)

[3.6 Using the Workflow Views 56](#_Toc30015790)

[3.7 Copying and Pasting Workflow Nodes 58](#_Toc30015791)

[3.8 Deleting Nodes 60](#_Toc30015792)

[3.9 Using the Node Tree 61](#_Toc30015793)

[3.10 Using the Overview Pane 63](#_Toc30015794)

[3.11 Using the Node Properties View 64](#_Toc30015795)

[3.12 Lock / Unlock Function 64](#_Toc30015796)

[3.13 Using the Edit Node Properties Dialog 65](#_Toc30015797)

[3.13.1 Using the Action Parameters Tab 66](#_Toc30015798)

[3.14 Command Line Options 67](#_Toc30015799)

[3.15 Using Keyboard Shortcuts 67](#_Toc30015800)

[Chapter 4 Workflow Node and Handler Library 69](#_Toc30015801)

[4.1 Process Nodes, Rule Nodes, and Switch Nodes 69](#_Toc30015802)

[4.1.1 Default Workflow Node Persistence Setting 69](#_Toc30015803)

[4.1.2 Activate 72](#_Toc30015804)

[4.1.3 Add 76](#_Toc30015805)

[4.1.4 AppendToResourceList 77](#_Toc30015806)

[4.1.5 AppendToTaskList 78](#_Toc30015807)

[4.1.6 ArrayJsonToJavaList 80](#_Toc30015808)

[4.1.7 AskFor 81](#_Toc30015809)

[4.1.7.1 Form Presentation 84](#_Toc30015810)

[4.1.7.2 Creating Custom Forms 84](#_Toc30015811)

[4.1.7.3 Validator Interface 84](#_Toc30015812)

[4.1.8 Assign 86](#_Toc30015813)

[4.1.9 Audit 87](#_Toc30015814)

[4.1.10 ChangeRoles 90](#_Toc30015815)

[4.1.11 ComposeMessage 91](#_Toc30015816)

[4.1.12 ConcatenateTaskLists 94](#_Toc30015817)

[4.1.13 ConfirmResourceReservation 95](#_Toc30015818)

[4.1.14 CreateBean 96](#_Toc30015819)

[4.1.15 CreateInventory 97](#_Toc30015820)

[4.1.16 CreateTaskList 99](#_Toc30015821)

[4.1.17 DateConverter 101](#_Toc30015822)

[4.1.18 Decrypt 104](#_Toc30015823)

[4.1.19 DeleteCache 105](#_Toc30015824)

[4.1.20 DeleteInventory 106](#_Toc30015825)

[4.1.21 DeleteScheduledJob 107](#_Toc30015826)

[4.1.22 DeleteServiceInstance 108](#_Toc30015827)

[4.1.23 DoNothing 109](#_Toc30015828)

[4.1.24 Encrypt 110](#_Toc30015829)

[4.1.25 ExecuteMacro 111](#_Toc30015830)

[4.1.26 Equal 113](#_Toc30015831)

[4.1.27 ExecSQLQuery 114](#_Toc30015832)

[4.1.28 ExecSQLStatement 115](#_Toc30015833)

[4.1.29 ExecuteExternal 116](#_Toc30015834)

[4.1.30 ForEach 118](#_Toc30015835)

[4.1.31 GenericUIDialog 120](#_Toc30015836)

[4.1.31.1 Form Presentation 124](#_Toc30015837)

[4.1.31.2 Creating Custom Forms 124](#_Toc30015838)

[4.1.32 GetBaseFileName 125](#_Toc30015839)

[4.1.33 GetBeansNNMNode 126](#_Toc30015840)

[4.1.34 GetBusinessHoursAfterDuration 130](#_Toc30015841)

[4.1.35 GetCalendarTimezone 132](#_Toc30015842)

[4.1.36 GetNextIncludedTime 133](#_Toc30015843)

[4.1.37 GetTimeRangesOfBusinessDay 135](#_Toc30015844)

[4.1.38 GetOperatingSystem 137](#_Toc30015845)

[4.1.39 GreaterThan 138](#_Toc30015846)

[4.1.40 GreaterThanOrEqual 139](#_Toc30015847)

[4.1.41 HTTPGet 140](#_Toc30015848)

[4.1.42 HTTPRequest 142](#_Toc30015849)

[4.1.43 InsertIntoTasklist 144](#_Toc30015850)

[4.1.44 InvokeInventoryMethod 145](#_Toc30015851)

[4.1.45 InvokeMethod 146](#_Toc30015852)

[4.1.46 IsModuleConfigured 148](#_Toc30015853)

[4.1.47 IsTimeIncluded 149](#_Toc30015854)

[4.1.48 IsTrue 151](#_Toc30015855)

[4.1.49 JavaToJson 152](#_Toc30015856)

[4.1.51 JsonToJava 153](#_Toc30015857)

[4.1.52 Java 154](#_Toc30015858)

[4.1.53 JavaRule 158](#_Toc30015859)

[4.1.54 JavaSwitch 160](#_Toc30015860)

[4.1.55 KillJob 162](#_Toc30015861)

[4.1.56 LessThan 163](#_Toc30015862)

[4.1.57 LessThanOrEqual 164](#_Toc30015863)

[4.1.58 Log 165](#_Toc30015864)

[4.1.59 MailRequest 166](#_Toc30015865)

[4.1.60 MapData 168](#_Toc30015866)

[4.1.61 MatchDBQuery 170](#_Toc30015867)

[4.1.62 MatchDBStore 172](#_Toc30015868)

[4.1.63 MethodRule 174](#_Toc30015869)

[4.1.64 MethodSwitch 175](#_Toc30015870)

[4.1.65 ModifyScheduledJob 176](#_Toc30015871)

[4.1.66 MoveFile 178](#_Toc30015872)

[4.1.67 MultiAssign 179](#_Toc30015873)

[4.1.68 Multiply 180](#_Toc30015874)

[4.1.69 MutexGetInfo 181](#_Toc30015875)

[4.1.70 MutexLock 182](#_Toc30015876)

[4.1.71 MutexSetInfo 184](#_Toc30015877)

[4.1.72 MutexUnlock 185](#_Toc30015878)

[4.1.73 Not 186](#_Toc30015879)

[4.1.74 PAYG 187](#_Toc30015880)

[4.1.75 ParseTemplate 188](#_Toc30015881)

[4.1.76 PatternMatch 189](#_Toc30015882)

[4.1.77 PPU 190](#_Toc30015883)

[4.1.78 PutMessage 191](#_Toc30015884)

[4.1.79 QueryInventory 194](#_Toc30015885)

[4.1.80 QueryScheduledJob 196](#_Toc30015886)

[4.1.81 QueryServiceInstance 198](#_Toc30015887)

[4.1.82 QueryServiceInstanceAll 200](#_Toc30015888)

[4.1.83 RandomInteger 201](#_Toc30015889)

[4.1.84 ReadData 202](#_Toc30015890)

[4.1.85 ReadDataFromDatabase 205](#_Toc30015891)

[4.1.86 ReadFile 207](#_Toc30015892)

[4.1.87 ReceiveMessage 208](#_Toc30015893)

[4.1.88 ReceiveMessageAndWait 209](#_Toc30015894)

[4.1.89 RediscoverHostsNNMNode 210](#_Toc30015895)

[4.1.90 ReleaseResource 210](#_Toc30015896)

[4.1.91 RemoveData 212](#_Toc30015897)

[4.1.92 RemoveFile 214](#_Toc30015898)

[4.1.93 Replace 215](#_Toc30015899)

[4.1.94 ReserveResource 216](#_Toc30015900)

[4.1.95 RetrieveSequence 219](#_Toc30015901)

[4.1.96 ScheduleCurrentJob 220](#_Toc30015902)

[4.1.97 ScheduleJob 221](#_Toc30015903)

[4.1.98 SendAlarm 223](#_Toc30015904)

[4.1.99 SendMail 225](#_Toc30015905)

[4.1.100 SendMessage 227](#_Toc30015906)

[4.1.101 SendSNMPTrap 230](#_Toc30015907)

[4.1.102 Sleep 231](#_Toc30015908)

[4.1.103 SORGetData 232](#_Toc30015909)

[4.1.104 SORSaveMessage 233](#_Toc30015910)

[4.1.105 StartJob 235](#_Toc30015911)

[4.1.106 StartJobAndWait 237](#_Toc30015912)

[4.1.107 StartQueueJob 239](#_Toc30015913)

[4.1.108 Switch 241](#_Toc30015914)

[4.1.109 Sync 242](#_Toc30015915)

[4.1.110 ThrowError 244](#_Toc30015916)

[4.1.111 ThrowException 245](#_Toc30015917)

[4.1.112 ThrowRuntimeException 246](#_Toc30015918)

[4.1.113 TransformXML 247](#_Toc30015919)

[4.1.114 UpdateBean 252](#_Toc30015920)

[4.1.115 UpdateCustomAttributesNNMNode 253](#_Toc30015921)

[4.1.116 UpdateInProgress 254](#_Toc30015922)

[4.1.117 UpdateInventory 255](#_Toc30015923)

[4.1.118 UpdateManagementModeNNMNode 257](#_Toc30015924)

[4.1.119 UpdateServiceInstance 257](#_Toc30015925)

[4.1.120 VariableMapper 259](#_Toc30015926)

[4.1.121 WasPreviousNodeOK 261](#_Toc30015927)

[4.1.122 WriteCasePacket 262](#_Toc30015928)

[4.1.123 WriteDataToDatabase 263](#_Toc30015929)

[4.1.124 WSComposeRequest 265](#_Toc30015930)

[4.1.124.1 Custom handlers 267](#_Toc30015931)

[4.1.125 WSRequest 269](#_Toc30015932)

[4.1.126 WriteFile 272](#_Toc30015933)

[4.1.127 XMLMapper 273](#_Toc30015934)

[4.1.127.1 XML Namespaces 276](#_Toc30015935)

[4.1.128 XMLParser 277](#_Toc30015936)

[4.2 Handlers 281](#_Toc30015937)

[4.2.1 ComposeMessageHandler 282](#_Toc30015938)

[4.2.2 MultiAssignHandler 283](#_Toc30015939)

[4.2.3 DoNothingHandler 284](#_Toc30015940)

[4.2.4 MethodHandler 285](#_Toc30015941)

[4.2.5 PutMessageHandler 286](#_Toc30015942)

[4.2.6 ReleaseResourceHandler 287](#_Toc30015943)

[4.2.7 QueueNotificationModuleHandler 288](#_Toc30015944)

[4.2.8 SendMessageHandler 289](#_Toc30015945)

[4.2.9 SORSaveMessageHandler 291](#_Toc30015946)

[4.2.10 SyncHandler 293](#_Toc30015947)

[4.2.11 VariableMapperHandler 294](#_Toc30015948)

[Chapter 5 Configuring the Workflow Manager 295](#_Toc30015949)

[5.1 Setting the Workflow Manager Parameters 295](#_Toc30015950)

[5.2 Understanding Workflow Manager Modules 299](#_Toc30015951)

[5.2.1 Required and Typical Workflow Manager Modules 299](#_Toc30015952)

[5.2.1.1 Logging 299](#_Toc30015953)

[5.2.1.2 Work Manager 300](#_Toc30015954)

[5.2.1.3 Transactional State 300](#_Toc30015955)

[5.2.1.4 Activation 300](#_Toc30015956)

[5.2.1.5 Authentication 300](#_Toc30015957)

[5.2.1.6 Database Access 301](#_Toc30015958)

[5.2.1.7 Parent-Child Synchronization 301](#_Toc30015959)

[5.2.1.8 Receiving Messages 301](#_Toc30015960)

[5.2.1.9 Sending Messages 301](#_Toc30015961)

[5.2.1.10 Keep Alive 302](#_Toc30015962)

[5.2.1.11 Distribution 302](#_Toc30015963)

[Chapter 6 Workflow Manager Module Library 303](#_Toc30015964)

[6.1 Using the Workflow Manager Module Library 303](#_Toc30015965)

[6.2 ActivationModule 304](#_Toc30015966)

[6.3 AuditModule 306](#_Toc30015967)

[6.4 BusinessCalendarModule 309](#_Toc30015968)

[6.5 Cache Module 310](#_Toc30015969)

[6.6 CasePacketDistModule 311](#_Toc30015970)

[6.6.1 Request to start a job with a set of initial case-packets 311](#_Toc30015971)

[6.6.2 Request to start a job without a set of initial case-packets 311](#_Toc30015972)

[6.7 CheckTimeModule 313](#_Toc30015973)

[6.8 ConflictModule 314](#_Toc30015974)

[6.9 DatabaseAdvancedAuthModule 315](#_Toc30015975)

[6.10 DBTransactionModule 318](#_Toc30015976)

[6.11 DelegatedAuthModule 319](#_Toc30015977)

[6.11.1 Architecture 319](#_Toc30015978)

[6.11.1.1 SAML 319](#_Toc30015979)

[6.11.1.2 OIDC 320](#_Toc30015980)

[6.11.1.3 Security 320](#_Toc30015981)

[6.11.1.4 Roles 321](#_Toc30015982)

[6.11.2 Module configuration 322](#_Toc30015983)

[6.11.2.1 Internal user cache 327](#_Toc30015984)

[6.11.3 Module configuration examples 327](#_Toc30015985)

[OIDC + HTTPS 327](#_Toc30015986)

[OIDC + HTTP 328](#_Toc30015987)

[SAML + encrypted assertions 328](#_Toc30015988)

[SAML 328](#_Toc30015989)

[6.11.4 Wildfly configuration 329](#_Toc30015990)

[6.11.5 Internal Authentication API 331](#_Toc30015991)

[6.11.6 IdP Requests 334](#_Toc30015992)

[6.11.6.1 SAML 334](#_Toc30015993)

[6.11.6.2 OIDC 335](#_Toc30015994)

[6.11.7 Token examples 336](#_Toc30015995)

[6.12 HTTPRequestModule 343](#_Toc30015996)

[6.13 HTTPSenderModule 345](#_Toc30015997)

[6.14 JMSListenerModule 348](#_Toc30015998)

[6.15 JMSSenderModule 352](#_Toc30015999)

[6.16 JNDIDatabaseModule 354](#_Toc30016000)

[6.17 KeepAliveModule 355](#_Toc30016001)

[6.18 LDAPAuthModule 358](#_Toc30016002)

[6.19 LinuxAdvancedAuthModule 362](#_Toc30016003)

[6.20 LoadFactorDistModule 364](#_Toc30016004)

[6.21 LogSearchModule 366](#_Toc30016005)

[6.22 MailHook 368](#_Toc30016006)

[6.23 MailRequestModule 369](#_Toc30016007)

[6.24 MailSenderModule 371](#_Toc30016008)

[6.25 Monitor 373](#_Toc30016009)

[6.26 NNMRequestModule 374](#_Toc30016010)

[6.27 OVOMessageModule 376](#_Toc30016011)

[6.28 QueueDistModule 377](#_Toc30016012)

[6.29 QueueDBPersistenceModule 378](#_Toc30016013)

[6.30 QueueModule 379](#_Toc30016014)

[6.31 QueueNotificationModule 380](#_Toc30016015)

[6.32 RoundRobinDistModule 384](#_Toc30016016)

[6.33 SchedulerModule 385](#_Toc30016017)

[6.34 SelfMonitoringModule 387](#_Toc30016018)

[6.35 ServiceOrderRegistryModule 389](#_Toc30016019)

[6.36 ServiceOrderRegistryMonitoringModule 391](#_Toc30016020)

[6.37 SNMPSenderModule 393](#_Toc30016021)

[6.38 SocketListenerModule 395](#_Toc30016022)

[6.39 SocketSenderModule 398](#_Toc30016023)

[6.40 SolutionXMLLogModule 400](#_Toc30016024)

[6.41 SyncModule 402](#_Toc30016025)

[6.42 UsageMonitoringModule 405](#_Toc30016026)

[6.43 WindowsAdvancedAuthModule 406](#_Toc30016027)

[6.44 WorkManagerModule 408](#_Toc30016028)

[6.45 XMLLogModule 409](#_Toc30016029)

[6.46 WSModule 411](#_Toc30016030)

[6.46.1 WSDescriptor 413](#_Toc30016031)

[6.46.2 Dependency management 415](#_Toc30016032)

[6.46.3 Fault and exception management 415](#_Toc30016033)

[Chapter 7 Writing Custom Workflow Nodes 416](#_Toc30016034)

[7.1 Understanding Workflow Nodes 416](#_Toc30016035)

[7.1.1 Accessing Workflow Manager Capabilities: WFContext & WFManager 417](#_Toc30016036)

[7.1.2 Example Source Code for Nodes 417](#_Toc30016037)

[7.1.3 Writing Custom Process Nodes 417](#_Toc30016038)

[7.1.3.1 init() Method 418](#_Toc30016039)

[7.1.3.2 nodeEntered() Method 418](#_Toc30016040)

[7.1.3.3 nodeExited() Method 418](#_Toc30016041)

[7.1.4 Writing Custom Rule Nodes 420](#_Toc30016042)

[7.1.4.1 init() Method 421](#_Toc30016043)

[7.1.4.2 eval() Method 421](#_Toc30016044)

[7.1.5 Writing Custom Switch Nodes 422](#_Toc30016045)

[7.1.5.1 init() Method 422](#_Toc30016046)

[7.1.5.2 evalKey() Method 423](#_Toc30016047)

[7.1.6 Writing Error and End Handlers 423](#_Toc30016048)

[7.2 Deploying Workflow Nodes and Handlers 424](#_Toc30016049)

[7.2.1 Using Custom Nodes and Handlers in Designer 424](#_Toc30016050)

[7.2.1.1 Component Descriptor Vocabulary 424](#_Toc30016051)

[7.2.1.2 DTD Quick Reference 425](#_Toc30016052)

[Chapter 8 Writing New Workflow Modules 427](#_Toc30016053)

[8.1 Writing New Workflow Manager Modules 427](#_Toc30016054)

[8.1.1 Example Source Code for Modules 427](#_Toc30016055)

[8.1.2 Implementation of Modules 427](#_Toc30016056)

[8.1.2.1 init Method 428](#_Toc30016057)

[8.1.2.2 shutdown Method 428](#_Toc30016058)

[8.1.2.3 isActive Method 428](#_Toc30016059)

[8.1.2.4 removeJob Method 428](#_Toc30016060)

[8.1.2.5 reconfigure Method 428](#_Toc30016061)

[8.1.2.6 suspend Method 429](#_Toc30016062)

[8.1.2.7 resume Method 429](#_Toc30016063)

[8.1.2.8 locked Method 429](#_Toc30016064)

[8.1.2.9 unlocked Method 429](#_Toc30016065)

[8.1.2.10 nodeDown Method 429](#_Toc30016066)

[8.1.2.11 nodeUp Method 430](#_Toc30016067)

[8.1.2.12 nodeLocked Method 430](#_Toc30016068)

[8.1.2.13 nodeUnlocked Method 430](#_Toc30016069)

[8.1.2.14 nodeSuspended Method 430](#_Toc30016070)

[8.1.2.15 nodeResumed Method 430](#_Toc30016071)

[8.1.2.16 takeover Method 431](#_Toc30016072)

[8.1.2.17 refresh Method 431](#_Toc30016073)

[8.1.2.18 discard Method 431](#_Toc30016074)

[8.1.3 Master-Slave 431](#_Toc30016075)

[8.1.3.1 Step 1 432](#_Toc30016076)

[8.1.3.2 Step 2 432](#_Toc30016077)

[8.1.3.3 Step 3 432](#_Toc30016078)

[8.1.3.4 Step 4 432](#_Toc30016079)

[8.1.3.5 Step 5 432](#_Toc30016080)

[8.1.3.6 Step 6 432](#_Toc30016081)

[8.1.4 Writing New Authenticator Module 433](#_Toc30016082)

[8.1.4.1 AuthModule methods 433](#_Toc30016083)

[8.1.4.2 AdvancedAuthModule methods 433](#_Toc30016084)

[8.1.4.3 Role Mapping 433](#_Toc30016085)

[8.1.4.4 Role Mapping Interface 433](#_Toc30016086)

[8.1.4.5 User Management Interface 433](#_Toc30016087)

[8.1.5 Writing New Queue Hook 435](#_Toc30016088)

[8.1.5.1 QueueHook Methods 436](#_Toc30016089)

[8.1.6 Writing New Sender Module 436](#_Toc30016090)

[8.1.6.1 SenderModule Methods 436](#_Toc30016091)

[8.1.7 Writing New Message Module 436](#_Toc30016092)

[8.1.7.1 MessageModule Methods 436](#_Toc30016093)

[8.2 Deploying Workflow Manager Modules 437](#_Toc30016094)

[Chapter 9 Writing Workflow Manager Clients 438](#_Toc30016095)

[9.1 Writing Workflow Manager External Interface Clients 438](#_Toc30016096)

[9.2 Creating a Workflow Manager Client 438](#_Toc30016097)

[9.3 Examples 439](#_Toc30016098)

[Appendix A mwfmtool 441](#_Toc30016099)

[A.1 mwfmtool 441](#_Toc30016100)

[A.1.1 Start mwfmtool 441](#_Toc30016101)

[A.1.2 Using mwfmtool from a Remote Computer 442](#_Toc30016102)

[A.1.3 Move mwfmtool to a Computer Running UNIX 442](#_Toc30016103)

[A.1.4 Move mwfmtool to a Computer Running Microsoft Windows 443](#_Toc30016104)

[A.1.5 mwfmtool Commands 443](#_Toc30016105)

[Appendix B Creating Additional Data Source 452](#_Toc30016106)

# List of Tables

[Table 2‑1 Default Case-packet Variables 34](#_Toc30016107)

[Table 2‑2 Examples of Complex Data Type Access 40](#_Toc30016108)

[Table 3‑1 Process Nodes 50](#_Toc30016109)

[Table 3‑2 Workflow Designer – Keyboard Shortcuts 67](#_Toc30016110)

[Table 4‑1 Default Workflow Node Persistence Setting 69](#_Toc30016111)

[Table 4‑2 Activate Parameters 73](#_Toc30016112)

[Table 4‑3 Add Parameters 76](#_Toc30016113)

[Table 4‑4 AppendToResourceList Parameters 77](#_Toc30016114)

[Table 4‑5 AppendToTaskList Parameters 78](#_Toc30016115)

[Table 4‑6 ArrayJsonToJavaList Parameters 80](#_Toc30016116)

[Table 4‑7 AskFor Parameters 82](#_Toc30016117)

[Table 4‑8 Assign Parameters 86](#_Toc30016118)

[Table 4‑9 Audit Parameters 87](#_Toc30016119)

[Table 4‑10 ChangeRoles Parameters 90](#_Toc30016120)

[Table 4‑11 ComposeMessage Parameters 91](#_Toc30016121)

[Table 4‑12 ConcatenateTaskList Parameters 94](#_Toc30016122)

[Table 4‑13 ConfirmResourceReservation Parameters 95](#_Toc30016123)

[Table 4‑14 CreateBean Parameters 96](#_Toc30016124)

[Table 4‑15 CreateInventory Parameters 97](#_Toc30016125)

[Table 4‑16 CreateTaskList Parameters 99](#_Toc30016126)

[Table 4‑17 DateConverter Parameters 101](#_Toc30016127)

[Table 4‑18 Decrypt Parameters 104](#_Toc30016128)

[Table 4‑19 DeleteCache Parameters 105](#_Toc30016129)

[Table 4‑20 DeleteInventory Parameters 106](#_Toc30016130)

[Table 4‑21 DeleteScheduledJob Parameters 107](#_Toc30016131)

[Table 4‑22 DeleteServiceInstance Parameters 108](#_Toc30016132)

[Table 4‑23 DoNothing Parameters 109](#_Toc30016133)

[Table 4‑24 Encrypt Parameters 110](#_Toc30016134)

[Table 4‑25 ExecuteMacreo Parameters 111](#_Toc30016135)

[Table 4‑26 Equal Parameters 113](#_Toc30016136)

[Table 4‑27 ExecSQLQuery Parameters 114](#_Toc30016137)

[Table 4‑28 ExecSQLStatement Parameters 115](#_Toc30016138)

[Table 4‑29 ExecuteExternal Parameters 116](#_Toc30016139)

[Table 4‑30 ForEach Parameters 118](#_Toc30016140)

[Table 4‑31 GenericUIDialog Parameters 121](#_Toc30016141)

[Table 4‑32 GetBaseFileName Parameters 125](#_Toc30016142)

[Table 4‑33 GetBeansNNMNode Available Attributes 126](#_Toc30016143)

[Table 4‑34 GetBeansNNMNode Parameters 127](#_Toc30016144)

[Table 4‑35 GetBusinessHoursAfterDuration Parameters 130](#_Toc30016145)

[Table 4‑36 GetCalendarTimezone Parameters 132](#_Toc30016146)

[Table 4‑37 GetNextIncludedTime Parameters 133](#_Toc30016147)

[Table 4‑38 GetTimeRangesOfBusinessDay Parameters 135](#_Toc30016148)

[Table 4‑39 GetOperatingSystem Parameters 137](#_Toc30016149)

[Table 4‑40 GreaterThan Parameters 138](#_Toc30016150)

[Table 4‑41 GreaterThanOrEqual Parameters 139](#_Toc30016151)

[Table 4‑42 HTTPGet Parameters 140](#_Toc30016152)

[Table 4‑43 HTTPRequest Parameters 142](#_Toc30016153)

[Table 4‑44 InsertIntoTasklist Parameters 144](#_Toc30016154)

[Table 4‑45 InvokeInventoryMethod Parameters 145](#_Toc30016155)

[Table 4‑46 InvokeMethod – Conversion to Boolean Type 146](#_Toc30016156)

[Table 4‑47 InvokeMethod Parameters 146](#_Toc30016157)

[Table 4‑48 IsModuleConfigured Parameters 148](#_Toc30016158)

[Table 4‑49 IsTimeIncluded Parameters 149](#_Toc30016159)

[Table 4‑50 IsTrue Parameters 151](#_Toc30016160)

[Table 4‑51 JavaToJson Parameters 152](#_Toc30016161)

[Table 4‑52 JsonToJava Parameters 153](#_Toc30016162)

[Table 4‑53 Java Parameters 155](#_Toc30016163)

[Table 4‑54 JavaRule Parameters 158](#_Toc30016164)

[Table 4‑55 JavaSwitch Parameters 160](#_Toc30016165)

[Table 4‑56 KillJob Parameters 162](#_Toc30016166)

[Table 4‑57 LessThan Parameters 163](#_Toc30016167)

[Table 4‑58 LessThanOrEqual Parameters 164](#_Toc30016168)

[Table 4‑59 Log Parameters 165](#_Toc30016169)

[Table 4‑60 MailRequest Parameters 166](#_Toc30016170)

[Table 4‑61 MapData Parameters 168](#_Toc30016171)

[Table 4‑62 MatchDBQuery Parameters 170](#_Toc30016172)

[Table 4‑63 MatchDBStore Parameters 172](#_Toc30016173)

[Table 4‑64 MethodRule Parameters 174](#_Toc30016174)

[Table 4‑65 MethodSwitch Parameters 175](#_Toc30016175)

[Table 4‑66 ModifyScheduleJob Parameters 176](#_Toc30016176)

[Table 4‑67 MoveFile Parameters 178](#_Toc30016177)

[Table 4‑68 MultiAssign Parameters 179](#_Toc30016178)

[Table 4‑69 Multiply Parameters 180](#_Toc30016179)

[Table 4‑70 MutexGetInfo Parameters 181](#_Toc30016180)

[Table 4‑71 MutexLock Parameters 182](#_Toc30016181)

[Table 4‑72 MutexSetInfo Parameters 184](#_Toc30016182)

[Table 4‑73 MutexUnlock Parameters 185](#_Toc30016183)

[Table 4‑74 Not Parameters 186](#_Toc30016184)

[Table 4‑75 PAYG Parameters 187](#_Toc30016185)

[Table 4‑76 ParseTemplate Parameters 188](#_Toc30016186)

[Table 4‑77 PatternMatch Parameters 189](#_Toc30016187)

[Table 4‑78 PPU Parameters 190](#_Toc30016188)

[Table 4‑79 PutMessage Parameters 191](#_Toc30016189)

[Table 4‑80 QueryInventory Parameters 194](#_Toc30016190)

[Table 4‑81 QueryScheduledJob Parameters 196](#_Toc30016191)

[Table 4‑82 QueryServiceInstance Parameters 198](#_Toc30016192)

[Table 4‑83 QueryServiceInstanceAll Parameters 200](#_Toc30016193)

[Table 4‑84 RandomInteger Parameters 201](#_Toc30016194)

[Table 4‑85 ReadData Parameters 203](#_Toc30016195)

[Table 4‑86 ReadDataFromDatabase Parameters 205](#_Toc30016196)

[Table 4‑87 ReadFile Parameters 207](#_Toc30016197)

[Table 4‑88 ReceiveMessage Parameters 208](#_Toc30016198)

[Table 4‑89 ReceiveMessageAndWait Parameters 209](#_Toc30016199)

[Table 4‑90 RediscoverHostsNNMNode Parameters 210](#_Toc30016200)

[Table 4‑91 ReleaseResource Parameters 210](#_Toc30016201)

[Table 4‑92 RemoveData Parameters 212](#_Toc30016202)

[Table 4‑93 RemoveFile Parameters 214](#_Toc30016203)

[Table 4‑94 Replace Parameters 215](#_Toc30016204)

[Table 4‑95 ReserveResource Parameters 216](#_Toc30016205)

[Table 4‑96 RetrieveSequence Parameters 219](#_Toc30016206)

[Table 4‑97 ScheduleCurrentJob Parameters 220](#_Toc30016207)

[Table 4‑98 ScheduleJob Parameters 221](#_Toc30016208)

[Table 4‑99 SendAlarm Parameters 223](#_Toc30016209)

[Table 4‑100 SendEmail Parameters 225](#_Toc30016210)

[Table 4‑101 SendMessage Parameters 227](#_Toc30016211)

[Table 4‑102 SendSNMPTrap Parameters 230](#_Toc30016212)

[Table 4‑103 Sleep Parameters 231](#_Toc30016213)

[Table 4‑104 StartJob Parameters 235](#_Toc30016214)

[Table 4‑105 StartJobAndWait Parameters 237](#_Toc30016215)

[Table 4‑106 StartQueueJob Parameters 239](#_Toc30016216)

[Table 4‑107 Switch Parameters 241](#_Toc30016217)

[Table 4‑108 Sync Parameters 242](#_Toc30016218)

[Table 4‑109 ThrowError Parameters 244](#_Toc30016219)

[Table 4‑110 ThrowException Parameters 245](#_Toc30016220)

[Table 4‑111 ThrowRuntimeException Parameters 246](#_Toc30016221)

[Table 4‑112 TransformXML Parameters 247](#_Toc30016222)

[Table 4‑113 Complex Data Type Usage in XSLT 249](#_Toc30016223)

[Table 4‑114 UpdateBean Parameters 252](#_Toc30016224)

[Table 4‑115 UpdateCustomAttributesNNMNode Parameters 253](#_Toc30016225)

[Table 4‑116 UpdateInProgress Parameters 254](#_Toc30016226)

[Table 4‑117 UpdateInventory Parameters 255](#_Toc30016227)

[Table 4‑118 UpdateServiceInstance Parameters 257](#_Toc30016228)

[Table 4‑119 UpdateServiceInstance Parameters 257](#_Toc30016229)

[Table 4‑120 VariableMapper Parameters 259](#_Toc30016230)

[Table 4‑121 WasPreviousNodeOK Parameters 261](#_Toc30016231)

[Table 4‑122 WriteCasePacket Parameters 262](#_Toc30016232)

[Table 4‑123 WriteDataToDatabase Parameters 263](#_Toc30016233)

[Table 4‑124 WSComposeRequest Parameters 265](#_Toc30016234)

[Table 4‑125 WSRequest Parameters 269](#_Toc30016235)

[Table 4‑126 WriteFile Parameters 272](#_Toc30016236)

[Table 4‑127 XMLMapper Parameters 274](#_Toc30016237)

[Table 4‑128 XMLParser Parameters 279](#_Toc30016238)

[Table 4‑129 ComposeMessageHandler Parameters 282](#_Toc30016239)

[Table 4‑130 MultiAssignHandler Parameters 283](#_Toc30016240)

[Table 4‑131 DoNothingHandler Parameters 284](#_Toc30016241)

[Table 4‑132 MethodHandler Parameters 285](#_Toc30016242)

[Table 4‑133 PutMessageHandler Parameters 286](#_Toc30016243)

[Table 4‑134 ReleaseResourceHandler Parameters 287](#_Toc30016244)

[Table 4‑135 QueueNotificationModuleHandler case packet variables required. 288](#_Toc30016245)

[Table 4‑136 SendMessageHandler Parameters 289](#_Toc30016246)

[Table 4‑137 SyncHandler Parameters 293](#_Toc30016247)

[Table 4‑138 VariableMapperHandler Parameters 294](#_Toc30016248)

[Table 5‑1 Workflow Manager Parameters 295](#_Toc30016249)

[Table 6‑1 ActivationModule Parameters 304](#_Toc30016250)

[Table 6‑2 AuditModule Parameters 307](#_Toc30016251)

[Table 6‑3 CacheModule Parameters 310](#_Toc30016252)

[Table 6‑4 CasePacketDistModule Parameters 311](#_Toc30016253)

[Table 6‑5 CheckTimeModule Parameters 313](#_Toc30016254)

[Table 6‑6 ConflictModule Parameters 314](#_Toc30016255)

[Table 6‑7 DatabasedvancedAuthModule Parameters 315](#_Toc30016256)

[Table 6‑8 DelegatedAuthModule Parameters 322](#_Toc30016257)

[Table 6‑9 HTTPRequestModule Parameters 344](#_Toc30016258)

[Table 6‑10 HTTPSenderModule Parameters 346](#_Toc30016259)

[Table 6‑11 JMSListenerModule Parameters 348](#_Toc30016260)

[Table 6‑12 JMSSenderModule Parameters 352](#_Toc30016261)

[Table 6‑13 JNDIDatabaseModule Parameters 354](#_Toc30016262)

[Table 6‑14 KeepAlive Module Parameters 356](#_Toc30016263)

[Table 6‑15 LDAPAuthModule Parameters 359](#_Toc30016264)

[Table 6‑16 LinuxAdvancedAuthModule Parameters 362](#_Toc30016265)

[Table 6‑17 LoadFactorDistModule Parameters 364](#_Toc30016266)

[Table 6‑18 LogSearchModule Parameters 366](#_Toc30016267)

[Table 6‑19 MailHook Parameters 368](#_Toc30016268)

[Table 6‑20 MailRequestModule Parameters 369](#_Toc30016269)

[Table 6‑21 MailSenderModule Parameters 371](#_Toc30016270)

[Table 6‑22 Monitor Parameters 373](#_Toc30016271)

[Table 6‑23 NNMRequestModule Parameters 374](#_Toc30016272)

[Table 6‑24 OVOMessageModule Parameters 376](#_Toc30016273)

[Table 6‑25 QueueDistModule Parameters 377](#_Toc30016274)

[Table 6‑26 QueueNotificationModule Parameters 380](#_Toc30016275)

[Table 6‑27 RoundRobinDistModule Parameters 384](#_Toc30016276)

[Table 6‑28 SchedulerModule Parameters 385](#_Toc30016277)

[Table 6‑29 SelfMonitoringModule Parameters 387](#_Toc30016278)

[Table 6‑30 ServiceOrderRegistryModule Parameters 390](#_Toc30016279)

[Table 6‑31 ServiceOrderRegistryMonitoringModule Parameters 391](#_Toc30016280)

[Table 6‑32 SNMPSenderModule Parameters 393](#_Toc30016281)

[Table 6‑33 SocketListenerModule Parameters 396](#_Toc30016282)

[Table 6‑34 SocketSenderModule Parameters 398](#_Toc30016283)

[Table 6‑35 SolutionXMLLogModule Parameters 400](#_Toc30016284)

[Table 6‑36 SyncModule Parameters 404](#_Toc30016285)

[Table 6‑37 UsageMonitoringModule Parameters 405](#_Toc30016286)

[Table 6‑38 WindowsAdvancedAuthModule Parameters 406](#_Toc30016287)

[Table 6‑39 WorkManager Parameters 408](#_Toc30016288)

[Table 6‑40 XMLLogModule Parameters 409](#_Toc30016289)

[Table 6‑41 WSModule Parameters 411](#_Toc30016290)

[Table 9‑1 mwfmtool Files 442](#_Toc30016291)

[Table 9‑2 Command Structure 444](#_Toc30016292)

[Table 9‑3 Command List 444](#_Toc30016293)

# List of Figures

[Figure 3–1 Workflow Designer 49](#_Toc30016294)

[Figure 3–2 Workflow Designer Main Bars 50](#_Toc30016295)

[Figure 3–3 Rule Node 50](#_Toc30016296)

[Figure 3–4 Process Node 50](#_Toc30016297)

[Figure 3–5 Workflow Designer Main Menu 51](#_Toc30016298)

[Figure 3–6 Workflow Designer Main Utility Tool Box 53](#_Toc30016299)

[Figure 3–7 Workflow Designer Visual Properties Toolbox 53](#_Toc30016300)

[Figure 3–8 Context Sensitive Menu 55](#_Toc30016301)

[Figure 3–9 Workflow Drawing View 56](#_Toc30016302)

[Figure 3–10 handlers View 57](#_Toc30016303)

[Figure 3–11 Description View 57](#_Toc30016304)

[Figure 3–12 State View 58](#_Toc30016305)

[Figure 3–13 Workflow XML Code View 58](#_Toc30016306)

[Figure 3–14 Copy and Paste of Multiple Objects 59](#_Toc30016307)

[Figure 3–15 Paste Special Warning Dialog 59](#_Toc30016308)

[Figure 3–16 Workflow Nodes Copied to Microsoft Notepad 60](#_Toc30016309)

[Figure 3–17 Deleting a Single Node 61](#_Toc30016310)

[Figure 3–18 Deleting Multiple Nodes 61](#_Toc30016311)

[Figure 3–19 Workflow Node Tree 62](#_Toc30016312)

[Figure 3–20 Workflow Node Groups 63](#_Toc30016313)

[Figure 3–21 Workflow Overview Pane 63](#_Toc30016314)

[Figure 3–22 Workflow Designer Node Properties View – Node Attributes Tab 64](#_Toc30016315)

[Figure 3–23 Workflow Designer Node Properties View – Action Parameters tab 64](#_Toc30016316)

[Figure 3–24 Node Properties View with Locked Contents 65](#_Toc30016317)

[Figure 3–25 Workflow Designer Edit Node Properties Dialog (Node Attributes Tab) 65](#_Toc30016318)

[Figure 3–26 Workflow Designer Edit Node Properties Dialog (Action Parameters Tab) 66](#_Toc30016319)

[Figure 4–1 Creating a Task List at Run-Time 99](#_Toc30016320)

[Figure 7–1 Workflow Component Inheritance Hierarchy 416](#_Toc30016321)

# Install Location Descriptors

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

|  |  |
| --- | --- |
| Descriptor | What the Descriptor Represents |
| *$ACTIVATOR\_OPT* | The base install location of HPE Service Activator. The UNIX® location is /opt/OV/ServiceActivator The Windows® location is  <install drive>:\HP\OpenView\ServiceActivator |
| *$ACTIVATOR\_ETC* | The install location of specific HPE Service Activator files.  The UNIX location is /etc/opt/OV/ServiceActivator The Windows location is <install drive>:\HP\OpenView\ServiceActivator\etc |
| *$ACTIVATOR\_VAR* | The install location of specific HPE Service Activator files. The UNIX location is /var/opt/OV/ServiceActivator The Windows location is <install drive>:\HP\OpenView\ServiceActivator\var |
| *$ACTIVATOR\_BIN* | The install location of specific HPE Service Activator files. The UNIX location is /opt/OV/ServiceActivator/bin The Windows location is <install drive>:\HP\OpenView\ServiceActivator\bin |
| *$ACTIVATOR\_BIN* | The install location of specific HPE Service Activator files.  The UNIX location is /opt/OV/ServiceActivator/bin The Windows location is  <install drive>:\HP\OpenView\ServiceActivator\bin |
| *$JBOSS\_HOME* | The install location for JBoss. The UNIX location is /opt/HP/jboss The Windows location is <install drive>:\HP\jboss |
| *$JBOSS\_DEPLOY* | The install location of the HPE Service Activator J2EE components. The UNIX location is /opt/HP/jboss/standalone/deployments The Windows location is <install drive>:\HP\jboss\standalone\deployments |
| *$JBOSS\_EAR\_LIB* | Location for libraries (Java \*.jar files) to be executed by the HPE Service Activator engine (workflow manager and resource manager): *$JBOSS\_DEPLOY*/hpsa.ear/lib |
| *$JBOSS\_ACTIVATOR* | More specific location of HPE Service Activator UI components deployed in JBoss: *$JBOSS\_DEPLOY*/hpsa.ear/activator.war |
| *$ACTIVATOR\_VAR\_MAIL* | The location to hold email attachment for the specific HPE Service Activator mail modules. The UNIX location is /var/opt/OV/ServiceActivator/mail The Windows location is <install drive>:\HP\OpenView\ServiceActivator\var\mail |

# Conventions

The following typographical conventions are used in this guide.

|  |  |  |
| --- | --- | --- |
| Font | What the Font Represents | Example |
| *Italic* | Book or manual titles, and manpage names | Refer to the *HPE Service Activator—Workflows and the Workflow Manager* and the *Javadocs* for more information |
| Provides emphasis | You *must* follow these steps. |
| 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 |
| **Computer Bold** | Text that you must type | At the prompt, type: **ls –l** |

# In This Guide

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

# 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 HPE Service Activator to interact with many sources of information and perform various tasks.

There are tree 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 HPE 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,](#_bookmark36) Workflow Node and Handler Library. To create your own nodes, see Chapter 7, Writing Custom Workflow Nodes.

## 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 HPE 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, or in Workflow Manager configuration snippets. 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,](#_bookmark179) Configuring the Workflow Manager.

### 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 HPE Service Activator architecture.

You can also implement new Workflow Manager modules to extend the reach of HPE 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](#_bookmark180) for general information about modules. For more information on the Workflow Manager modules provided with HPE Service Activator, see “Using the Workflow Manager Module Library” on page 303. To create your own modules, see Chapter 8, "Writing New Workflow Modules”.

## 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 service inside WildFly.
* 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 usedby 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.

# Creating and Deploying Workflows

This chapter describes the structure of a workflow definition and explains how workflows are deployed in a HPE 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,](#_bookmark32) Using the Workflow Designer.

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

### General Structure of a 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 worklow will not be stored during its execution, which means that if HPE Service Activator is stopped while the workflow is running it will not be started again when HPE Service Activator is restarted. This will also be the case even if the transaction module is confiugred. The default value is “false”.

Here is an example of how the 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.

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** HPE 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”](#_bookmark185) and “Roles, Privileges and Authentication” in HPE 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 boble 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 HPE 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](#_bookmark36).

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 Procccess 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 Node

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.ReleaseResourcHandler

</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 HPE 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 HPE 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).

### 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 follow by zero of 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 |
| --- | --- | --- |
| RET\_VALUE | Integer | RET\_VALUE is a default case-packet variable, which is updated after each node execution. Each node has an internal attribute called ret\_value, 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. |
| RET\_TEXT | 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 ret\_text, 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. |
| EX\_STEP\_NAME | 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 determ what to do. |
| JOB\_ID | Integer | Unique identifier for each running job. You cannot modify this value from within a node. |
| STEP\_NAME | String | Unique name of the node currently being processed. You cannot modify this value from within a node. |
| HOST\_NAME | String | The name of the host where the job is running. You cannot modify this value from within a node. |
| WORKFLOW\_NAME | String | The name of the workflow the job is running. You cannot modify this value from within a node. |
| WORKFLOW\_VERSION | Integer | The workflow version. You cannot modify this value from within a node. |
| SOLUTION\_NAME | String | The Solution Name of the workflow. You cannot modify this value from within a node. |
| SUBSTEP | String | 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 WFContext.getSubstep() and WFContext.setSubstep() methods. The setter method ensures that the case-packet gets persisted when this value is set.  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 SUBSTEP variable to record the fact that the activation has actually been initiated. If the node is executed again and the SUBSTEP 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 continuous its work. |
| START\_TIME | Integer | Universal time coordinates (UTC) time when the the flow began. This value is maintained in the case of failures and cannot be modified from within a node. |
| TIMEOUT | Boolean | If true, indicates that a timeout occurred in a previous node (such as AskFor) while waiting for interaction. |
| PRIORITY | 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. This variable is used by e.g. the ActivationModule to determine the order in which items on the queue are processed. |
| DEFAULT\_ROLE | String | The workflow’s default role. |
| KILL\_ROLE | String | The workflow’s kill role. |
| TRACE\_ROLE | String | The workflow’s trace role |
| START\_ROLE | String | The workflow’s start role |
| STATUS | String | 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:   * Initted. The workflow has just been started. * Recovered. The workflow has just been recovered after a system shut down. * Running. The workflow is currently running code within a node. * Transit. The workflow is moving from one node to the next one. * Handling Error. An unexpected, uncaught exception was thrown from within a workflow node, and the error handler is currently running. * Waiting. The workflow is blocked in a node waiting for some input from a request queue. * Awakened. The workflow has just been awakened, unblocking it from the queue on which it was waiting. * Finishing. The end handler for this workflow is currently being run. |
| ETC | String | The full path to the $ACTIVATOR\_ETC directory. |
| VAR | String | The full path to the $ACTIVATOR\_VAR directory. |
| FILE\_URL\_PREFIX | 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:///. |
| SOLUTION\_ETC | String | The full path to the solution/etc directory. |
| SOLUTION\_VAR | String | The full path to the solution/var directory. |
| UNIQUE\_WORKFLOW | Integer | Indicateds if the workflow is unique or not. |
| SERVICE\_ID | String | The Service Id of the workflow. This varibale can be used to correlate a number of jobs to the same service. This varible must be set by the workflow to get a value. Per default it is empty. The varible 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 varibale can be used to correlate a number of jobs to the same order id. This varible must be set by the workflow to get a value. Per default it is empty. The varible 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 varible must be set by the workflow to get a value. Per default it is empty. The varible 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 varible must be set by the workflow to get a value. Per default it is empty. The varible is shown on the UI in a number of views e.g. the job view. |
| SCHEDULED\_INFO | Object | Used to save scheduled information. |
| THROW\_EXCEP | Boolean | This will set the behaviour for all the nodes in the workflow if the throw\_excep parameter is not set at the node level. Default value is true. |
| SCHEDULED\_INFO | 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 independet. |
| NULL | Object | Can be used by SI to assign a case-packet variable to a null value |
| activation\_major\_code | Integer | Required for use by the Activate node.  After the completion of an activation the Activate node will set this variable to hold the major\_code returned by the ExecutionDescriptor from the activation. |
| activation\_minor\_code | 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 minor\_code returned by the ExecutionDescriptor from the activation. |
| activation\_stdout | 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 stdout returned by the ExecutionDescriptor from the activation. |
| activation\_stderr | 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 stderr returned by the ExecutionDescriptor from the activation. |
| activation\_description | 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 description returned by the ExecutionDescriptor from the activation. |
| skip\_activation | 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 activation\_major\_code to either 0 or the current value of skip\_activation\_major\_code (if it exists), and it will set the activation\_minor\_code to either 0 or the current value of skip\_activation\_minor\_code (if it exists). |
| skip\_activation\_major\_code | Integer | Optional for use by the Activate node.  The Activate node will use this to set the activation\_major\_code if the value of skip\_activation is “true”. |
| skip\_activation\_minor\_code | Integer | Optional for use by the Activate node.  The Activate node will use this to set the activation\_minor\_code if the value of skip\_activation is “true”. |
| RUNTIME | Object | Optional variable containing a Map.  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.  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 RUNTIME variable.  Here is an example of accessing both the timestamp from the execution of the ConfirmActivationDetails node and the user who did the interaction:  RUNTIME{"ConfirmActivationDetails"}{"timestamp"}  RUNTIME{"ConfirmActivationDetails"}{"username"} |
| RESERVATIONS | Object | Optional variable containing an Array of Java Beans.  The ReserveResource, ReleaseResource, ConfirmResourceReservation nodes and ReleaseResourceHandler make use of this variable to record resources that have been reserved. See details in the description of these components. |
| message\_url | String | Optional variable containing the name of a file or database message id passed from the SocketListenerModule.  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. |
| module\_name | String | 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. |
| MUTEX | Boolean | Indicating if any mutexes ever have been locked in this job |
| DISABLE\_PERSISTENCE | Boolean | Can be included as part of the initial case packet when starting a workflow, to override the wokflow persistence settings of a persistent workflow. Cannot be used with executeMacro, and cannot be modified by the workflow. |

### References to Workflows 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 Java Bean 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 Java Bean or as an array of Java Beans.

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 Lists

If the case-packet variable contains an Array or a List, 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 nth 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.

#### Java Beans

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.propertyThis 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.key Index by a constant key value. This will cause the Workflow Manager to look for the method mapVarName.getKey(). If this method is not found, it will look for the method mapVarName.get(String key) and call that method if it exists. If none of these methods are found, and exception will be thrown.
* 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 2‑2 demonstrate this capability.

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

Table 2‑2 Examples of Complex Data Type Access

| Description | Syntax |
| --- | --- |
| Arrays indexed by constants | arrVarName[0] |
| Arrays indexed by variables | arrVarName[intVar] |
| Length of an array | arrVarName# |
| Arrays of arrays | arrVarName[0][1] |
| Fields of beans | beanVar.field |
| Arrays of beans | arrVarName[0].field |
| Elements of maps of maps | RUNTIME{"askForSwitchName"}{"timestamp"} |
| Fields of beans that are maps | beanVar.field{"key1"} |
| Arrays indexed by bean fields | arrVarName[beanVarName.field] |
| Vectors indexed by constants | vecVarName[4] |
| Vectors indexed by variables | vecVarName[intVar] |
| Size of a vector | vecVarName# |
| Number of elements in a HashMap (Map) | hashVarName# |
| Number of elements in a List (Collection) | listVarName# |
| Complex expressions | mapVarName{beanVar.field[intVar]} |
| {arrVarName[3]}.field# |
| vecVarName[3][intVar]["key2"] |

**NOTE** The MapData node offers an alternate method of extracting data from a HashMap. See“MapData” 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).

#### 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: 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" />

<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: 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 a 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"/>

<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: 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>

<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 Priorization in Workflows

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

#### Priorizing 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 PRIORITYdefault 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 PRIORITYcase-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.

##### Priorizing Activation Items

To use the Workflow Manager, you must configure an activation module. Class ActivationModule is supplied for this purpose.

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 Priorization 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 Priorization in Working Threads

Unless you have configured Min-Threadsdifferently, 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-Threadsvalue to 1. This will make a single worker thread available for working on workflows.
2. Restart the HPE 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 Priorization of Working 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\_threadsand max\_threadsset 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 PRIORITYto 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 304
* “WorkManagerModule” on page 408

### 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 HPE 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 72 for more information about making uploaded data available in a case-packet variable stored as a HashMap
* “Maps” on page 39 for more information about using the workflow syntax for accessing maps to extract uploaded data from a Map
* “MapData” on page 168 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 HPE 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 another 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.

# Using the Workflow Designer

The HPE 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 worlflow 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.

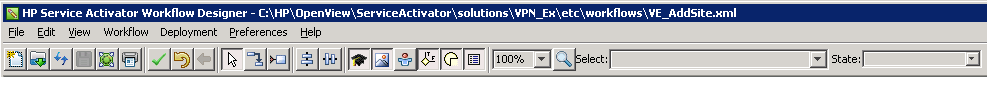
Figure 3–1 Workflow Designer



### 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:

Figure 3–2 Workflow Designer Main Bars



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

Table 3‑1 Process Nodes

|  |  |
| --- | --- |
| Node | Description |
|  | Basic layout of a process node; the name of the node is always displayed. |
|  | Process node displaying the node’s class name. |
|  | Process node displaying the node class name and icon. |
|  | Process node displaying the node icon. |
|  | Process node displaying the role. |
|  | Process node displaying the node class name and role. |
|  | Process node displaying the node class name, node icon, and role. |
|  | Process node displaying the node icon and role. |

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

Figure 3–3 Rule Node



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

Figure 3–4 Process Node



## Using the Main Menu

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

Figure 3–5 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:

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 then 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.

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.

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.

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

* Workflow Settings… Shows the Workflow Settings dialog, which contains four tabs:
  + 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 aswell 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.

* + Description Provides a large text area for composing a multi-line description of the workflow.
  + Roles Allows you to edit the roles that may carry out operations on this workflow.
* Add New Roles... Goes directly to the Roles tab of the Workflow Properties dialog for adding, deleting, and assigning workflow rules.
* Edit Case-Packet... Shows the Case-Packet dialog for adding, modifying and deleting case-packet variables, including setting the initial value of variables.
* Workflow Contract...Goes directly to the Workflow Contract dialog for adding and deleting case-packet variables to the contract.
* Persistency Allows you to enable or disable persistency. If persistence is enabled, the workflow data is stored in the database.

Deployment Deploying workflows to the database.

* Deploy Current Workflow Allows to deploy the workflow on which the user is currently working on.
* Deploy All Open Workflows Allows to deploy all the workflows that are opened or exist in the Workflow Designer UI.
* Deploy Workflows Allows to select and deploy workflows that exist in the file system.

First time deployment is done the user will be promt 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 remeber the values as long as it is running.

Preferences

* 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.

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 3–6 Workflow Designer Main Utility Tool Box



 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.

 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.

 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...

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

 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.

 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.

 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 3–7 Workflow Designer Visual Properties Toolbox



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

 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 CTRLkey 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.](#_bookmark35)
  + Context sensitive menu Right-click a node to get a context sensitive menu for that node.

 Draw Arrow Used to draw an arrow between two nodes:

* + - * Click the Draw Arrow icon
      * Click a source node (arrow starting node)
      * 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.

 Set Initial Node Click the Initial Node icon

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.

 Vertical Align or Horizontal Align

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

Select multiple nodes that are to be aligned. You may use any method to select multiple nodes.

Click the Vertical Align or Horizontal Align icon. All of the selected nodes will be moved to align with the first node selected.

  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.

 Zoom 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.

 Search When clicking this button a pop dialog is shown which makes it possible to enter different search criteria to searhc 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.

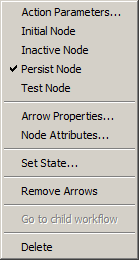
 SelectWorkflow Node Click and select one of the available workflow nodes.

 SelectWorkflow state 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 3–8 Context Sensitive Menu



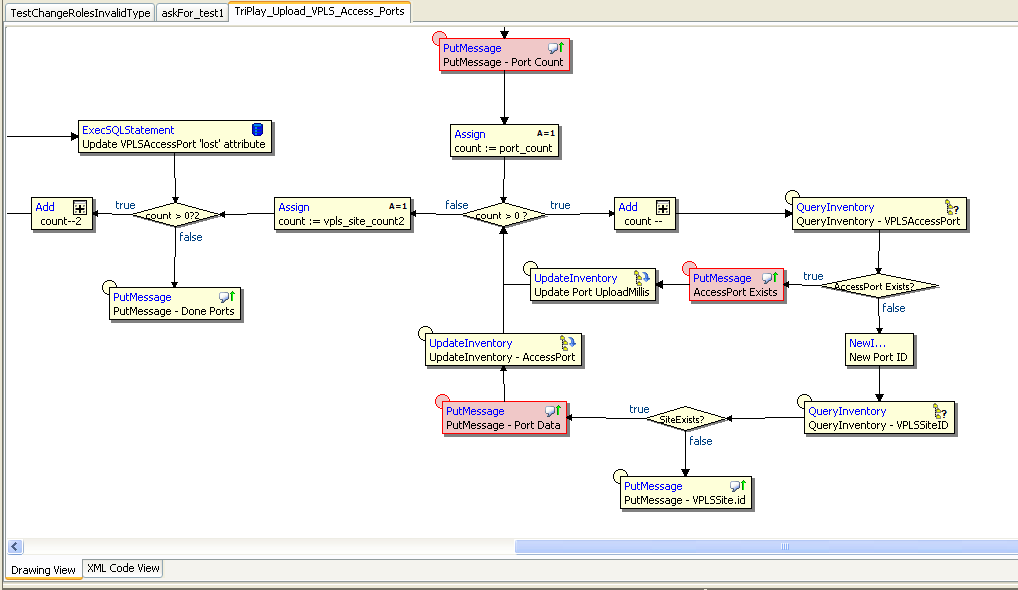
The menu contains the following actions:

* 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.
* Initial Node Set this node to be the starting node of the workflow.
* 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.
* 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.
* Persist Node Toggles whether persistence should be done after execution of the node.
* 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.
* 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.
* 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.
* 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 3–9 Workflow Drawing View

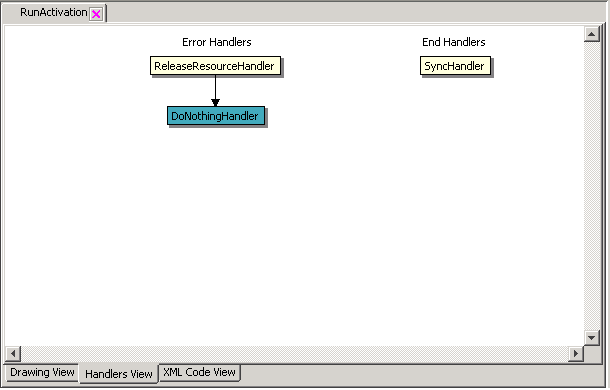


If a single node is selected while switching from Drawing View to XML Code View the Workflow Designer will scroll down to the XML definition of the selected node and the node name will become highlighted. An example of this is shown in Figure 3–13.

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

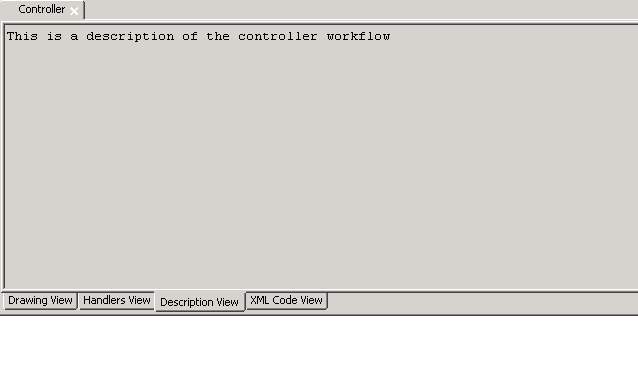
In the Handlers view the end and error handers which are defined will be shown in two different columns. If no handlers are defined the Handler View will be empty.

Figure 3–10 handlers View



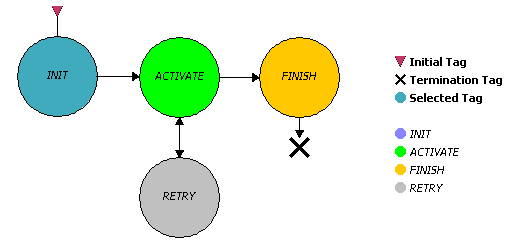
In the Description view the overall description of the workflow can be seen. The description field can be edited in the workflow settings.

Figure 3–11 Description View



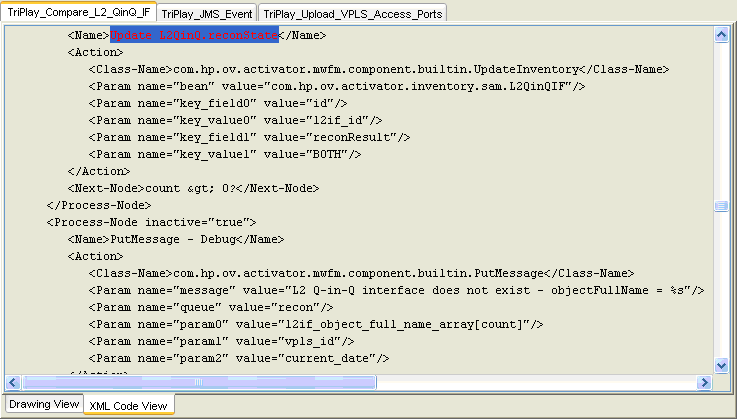
The state view shows the workflow represented as a (simplified) state diagram.

Figure 3–12 State View



The XML Code View show the xml content.

Figure 3–13 Workflow XML Code View



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.

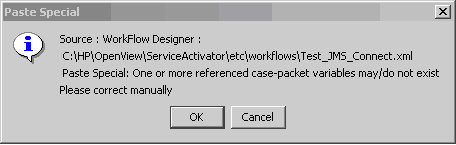
When you copy and paste multiple nodes the Workflow Designer will preserve all arrows for which the source as well as the destination nodes exists in the copy buffer; other arrows will be deleted. This is illustrated in Figure 3–14 where five nodes were copied and then pasted into the same workflow..

Figure 3–14 Copy and Paste of Multiple Objects



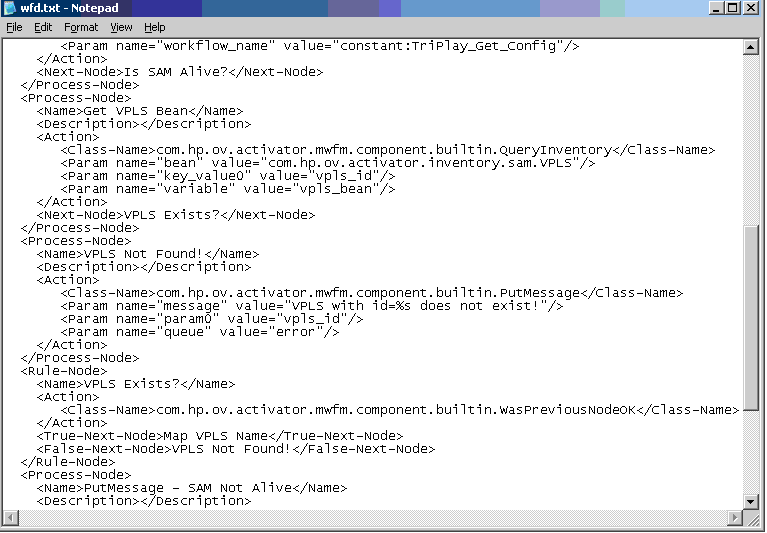
If you copy one or more workflow nodes in one workflow and paste them into another workflow by using Edit:Paste Special, the Workflow Designer will display a warning dialog; see Figure 3–15. The warning dialog is displayed to make it clear to the user that issues may occur if case-packet variables in the pasted nodes are in conflict with case-packet variables already existing in the destination workflow. For example, conflicting types.

Figure 3–15 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 3–16.

Figure 3–16 Workflow Nodes Copied to Microsoft Notepad

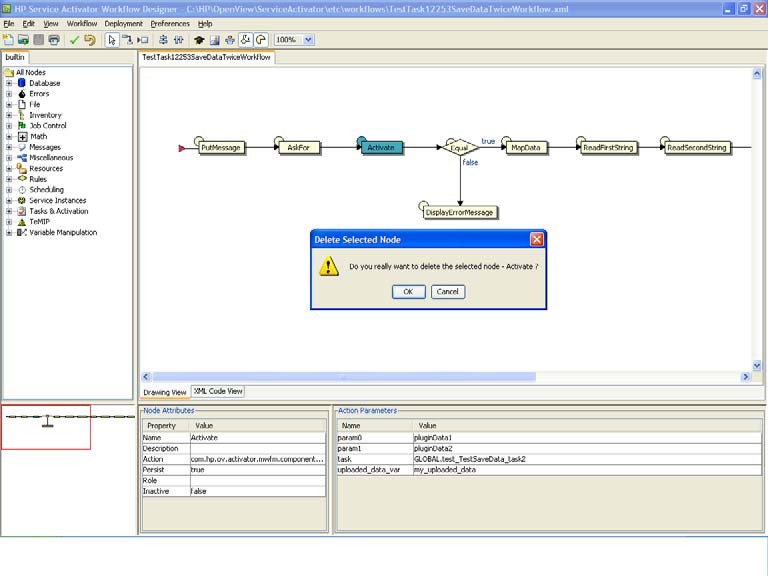


## Deleting Nodes

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.

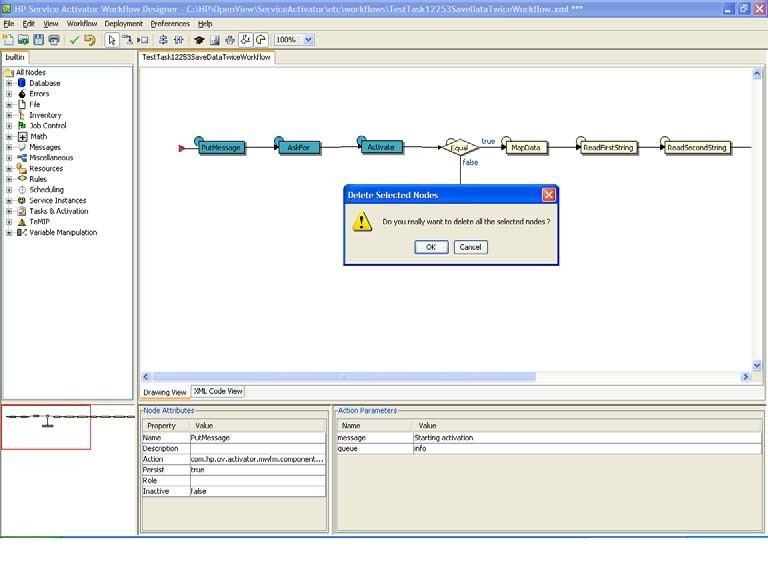
* Deleting a single node: In the Workflow Designer UI, you can select a single node and press the **Delete** key to delete the node.

Figure 3–17 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 3–18 Deleting Multiple Nodes

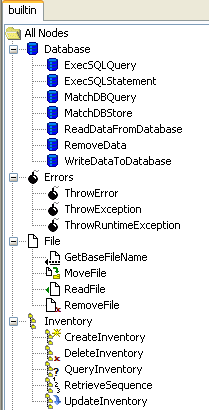


## Using the Node Tree

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 3–19 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.

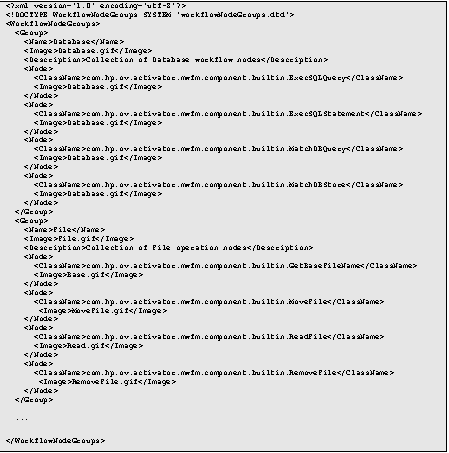
In each of these subdirectories the workflow nodes are grouped by their behavior (or by any other criteria decided by the user) based on the contents of the XML file workflowNodeGroups.xml which is located in the same directory as the workflow nodes.

**NOTE** The workflowNodeGroups.xml file is not mandatory. If there is no such file, all nodes will simply be show 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 3–20:

Figure 3–20 Workflow Node Groups



## Using the Overview Pane

Figure 3–21 Workflow Overview Pane

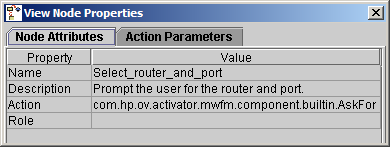


The workflow overview pane displays the entire workflow regardless of its size. At all times the overview pane shows a red box around the area of the workflow that is currently visible in the workflow view. This is particularly useful when you are editing a large workflow. Selected nodes and nodes marked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

**NOTE** It is possible to drag the red box in the workflow overview pane to navigate the workflow.

## Using the Node Properties View

Figure 3–22 Workflow Designer Node Properties View – Node Attributes Tab

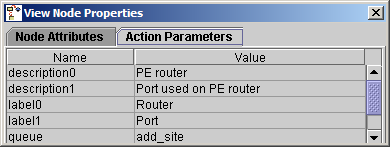


When you bring up a workflow in the design window, the node properties dialog box for that workflow is also displayed. This properties window can be used for reviewing the node properties, but also to change the properties. An other way to change the node properties is to double-click the node to bring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

The Node Attributes tab (shown in Figure 3–22) 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 3–23) 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 3–23 Workflow Designer Node Properties View – Action Parameters tab

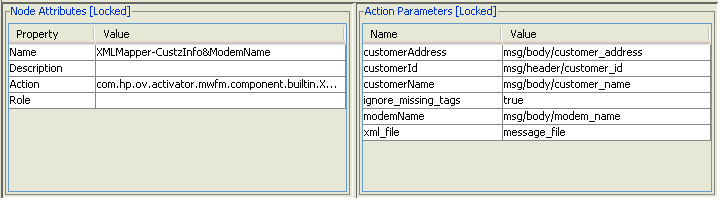


## Lock / Unlock Function

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 3–24.

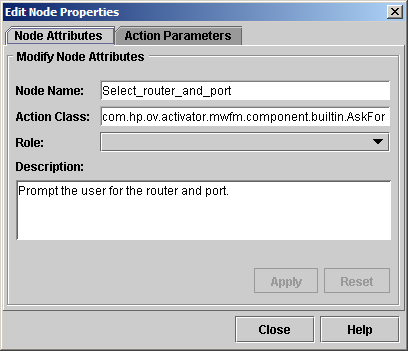
Figure 3–24 Node Properties View with Locked Contents



## 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 3–25 Workflow Designer Edit Node Properties Dialog (Node Attributes Tab)



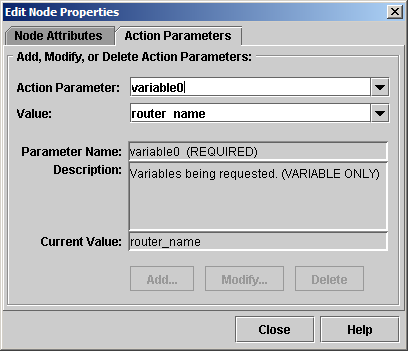
Use this tab to modify general node attributes.

* 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.
* 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 3–26 Workflow Designer Edit Node 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](#_bookmark34)).

**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.

Table 3‑2 Workflow Designer – Keyboard Shortcuts

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

# 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 HPE 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 HPE 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 Workflow Node Persistence Setting

| Node Name | Disable Persistence |
| --- | --- |
| Activate | FALSE |
| Add | TRUE |
| AppendToTaskList | TRUE |
| AppendToResourceList | TRUE |
| ArrayJsonToJava | TRUE |
| AskFor | FALSE |
| Assign | TRUE |
| Audit | FALSE |
| ChangeRoles | TRUE |
| ComposeMessage | TRUE |
| ConcatenateTaskLists | TRUE |
| ConfirmResourceReservation | TRUE |
| CreateInventory | FALSE |
| CreateBean | TRUE |
| CreateTaskList | TRUE |
| DateConverter | TRUE |
| DeleteCache | TRUE |
| DeleteInventory | FALSE |
| DeleteScheduleJob | FALSE |
| DeleteServiceInstance | FALSE |
| DoNothing | TRUE |
| 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 |
| GetTimeRangesOfBusinessDa y | TRUE |
| GreaterThan | TRUE |
| GreaterThanOrEqual | TRUE |
| HTTPGet | FALSE |
| HTTPRequest | FALSE |
| InsertIntoTaskList | TRUE |
| InvokeInventoryMethod | FALSE |
| InvokeMethod | FALSE |
| IsTimeIncluded | TRUE |
| IsTrue | TRUE |
| IsModule | TRUE |
| JsonToJava | TRUE |
| Java | FALSE |
| JavaRule | TRUE |
| JavaSwitch | TRUE |
| KillJob | FALSE |
| LessThan | TRUE |
| LessThanOrEqual | TRUE |
| Log | FALSE |
| MapData | TRUE |
| MatchDBQuery | TRUE |
| MatchDBStore | TRUE |
| MethodRule | FALSE |
| MethodSwitch | FALSE |
| ModifyScheduledJob | FALSE |
| MoveFile | FALSE |
| MultiAssign | TRUE |
| Multiply | TRUE |
| MutexGetInfo | TRUE |
| MutexLock | FALSE |
| MutexSetInfo | FALSE |
| MutexUnlock | FALSE |
| Not | TRUE |
| ParseTemplate | TRUE |
| PatternMatch | FALSE |
| PAYG | FALSE |
| PPU | FALSE |
| PutMessage | FALSE |
| QueryInventory | TRUE |
| QueryScheduledJobs | TRUE |
| QueryServiceInstance | TRUE |
| 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 |
| StartQueueJob | FALSE |
| Sync | FALSE |
| Switch | TRUE |
| ThrowError | FALSE |
| ThrowException | FALSE |
| ThrowRuntimeException | FALSE |
| TransformXML | TRUE |
| UpdateBean | TRUE |
| UpdateCustomAttributesNNMNode | FALSE |
| UpdateInProgress | FALSE |
| UpdateInventory | FALSE |
| UpdateServiceInstance | FALSE |
| VariableMapper | TRUE |
| WasPreviousNodeOK | TRUE |
| WriteCasePacket | FALSE |
| WriteDataToDatabase | FALSE |
| WriteFile | 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 99 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 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 304
* Section “Job Counters” in the document *HPE Service Activator User’s and Administrator’s Guide*

Table 4‑2 Activate Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| Task | Yes, if task\_list\_var is not used | A string indicating the name of a single atomic or compound task to execute.  This parameter is mutually exclusive with the task\_list\_var parameter. | None | String |
| param0 param1... paramN | Yes, if task 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 task\_list\_var parameter. | None | String |
| task\_list\_var | Yes, if task is not used | 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.  This parameter is mutually exclusive with the parameters task and param0, param1... paramN. | None | Object |
| activation\_module | No | Specifies the name of the activation module to use. If not specified, a module called activator is assumed. | “activator” | String |
| error\_queue | No | Specifies the name of a queue to which a message will be sent if the activation fails. | None | Queue |
| error\_message | No | Specifies the format of any error message that may be sent (if the error\_queue parameter is set). | None | String |
| ignore\_lock\_argument | No | When set to 'true' the resource manager will not do locking on the lock arguments (Default: false) | false | Boolean |
| uploaded\_data\_var | No | 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.  You only need to specify the uploaded\_data\_var 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. | None | Object |

**Example: 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: 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 |
| op0, op1 ... opN | Yes, at least one | Each operand (other than op0), can be a case-packet variable or a constant (specified as constant:X where X is the constant). op0 can only be a case-packet variable since that is where the result is stored. | None | Numeric |

**Example: 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 216

Table 4‑4 AppendToResourceList Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| resource\_list\_var | 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 |
| bean | Yes | Name of the Java class generated by Inventory Builder for a reservable resource from which to allocate the resource. | None | String |
| method | No | Name of the method by which the reservation is carried out. This defaults to doing a reservation using the reserveResoureByPrimaryKey method | reserve Resoure ByPrim aryKey | String |
| field | No | Name of the field by which the reservation is carried out.  This defaults to doing a reservation by PrimaryKey. | Primary Key | String |
| variable | 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 99 for more information about creating a new task list
* “ConcatenateTaskLists” on page 94
* “InsertIntoTasklist” on page 144
* “PutMessage” on page 191
* “Activate” on page 72 for more information about the Activate node

Table 4‑5 AppendToTaskList Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| task\_list\_var | Yes | A case-packet variable of type Object that contains the task list. Before tasks can be appended, the list must be created by CreateTaskList. | None | Object |
| task | 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 |
| param0 param1 ... paramN | 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: 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>

### ArrayJsonToJavaList

com.hp.ov.activator.mwfm.component.builtin.ArrayJsonToJavaList

The node will instanciate an Object from an array with JSONs.

Table 4‑6 ArrayJsonToJavaList Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| input\_json\_var | Yes | JSON text to be instantiated | None | String |
| output\_class | Yes | The class name of the Object to be created. | None | String |
| output\_var | Yes | Name of case packet variable to catch the Object that was created. | None | Object |

### 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.

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 242 for information about how to respond to an AskFor request from another workflow.
* “SyncModule” on page 402

Table 4‑7 AskFor Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| queue | 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 |
| variable0, variable1 ... variableN | Yes, at least one | 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 |
| 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 the 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 |
| timeout | No | The value can be a case-packet variable or a constant (specified as constant:*X* where *X* 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 |
| 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 |
| validation | No | The name of a Java class that implements the Validator interface. If you specify this class, the validate() 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 response parameter is ignored because the validate method returns a response message. | None | String |
| wait\_for\_child | 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 |
| children\_count | 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 |
| child\_workflow\_j ob\_id0, child\_workflow\_j ob\_id1 ... child\_workflow\_j ob\_idN | 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: 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 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>/<qu eue>.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” on page 81, 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 mssage 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 the are more flexible.

**See Also**

* “VariableMapper” on page 259
* “MultiAssign” on page 179

Table 4‑8 Assign Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| variable | Yes | Case-packet variable to be set. | None | String / Integer / Float / Boolean / Object |
| value | Yes | 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. |

**Example: 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 306 for more details on how to enable or disable event types.

Table 4‑9 Audit Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| audit\_module | No | Specifies the name of the audit module to use. If not specified, the module called “auditor” is assumed. | “auditor” | String |
| event\_type | 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 |
| timestamp | 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 *timestamp* is not specified, the current system time is used. The format for *timestamp* 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 |
| user | 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 case-packet variable 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 |
| step\_name | 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 |
| message | 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 |
| identifier | 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 |
| attrib\_name0 attrib\_name1… attrib\_nameN | 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 *attrib\_value* parameter. | None | String |
| attrib\_value0 attrib\_value1… attrib\_valueN | 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 |
| skip | No | The node will be skipped if the value is se to true. | false | Boolean |

**Example: 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>

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

</Class-Name>

<Param name="timestamp" value="RUNTIME{&quot;Add User&quot;}

{&quot;timestamp&qout;}"/>

<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 4‑10 ChangeRoles Parameters

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

**Example: 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 4‑11 ComposeMessage Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| template\_file | Yes, if template\_var 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 the directory *$ACTIVATOR\_ETC*/template\_files | None | String |
| template\_var0, template\_var1,  ... template\_varN | Yes, if template\_fileis not used | Name of a case-packets available that contains the template strings. | None | String |
| use\_solution\_dir | No | When set to "true", the nodes will read from *$SOLUTION\_ETC/*template\_files instead of $ACTIVATOR\_ETC/template\_files. | True | Boolean |
| output\_file | Yes, if output\_var is not used | Name of the file to which the composed message is to be written. The value of this parameter can be a case-packet variable that contains the name of the file, or it can be a constant (specified as constant:*X* where *X* is the name of the file).  If the path name to the file is not an absolute path, the file is created relative to *$ACTIVATOR\_VAR*/tmp | None | String |
| *output\_var* | Yes, if output\_file is not used | Name of a case-packet variable in which the composed message is placed. | None | String |

**Example: 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>

</Process-Node>

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

<response\_msg>

<header>

<message\_id>%message\_id%</message\_id>

<service\_id>%service\_id%</message\_id>

</header>

<body>

<message>OK</message>

</body>

</response\_msg>

During execution, the ComposeMessage node will substitute %message\_id% with the value of the message\_id case-packet variable.

**Example: ComposeMessage using multipe 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>

</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: ComposeMessage using multipe 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="messagetemplate"/>

<Param name="template\_var1 value="servicetemplate"/>

</Action>

</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 99 for more information about creating a new task list.
* “AppendToTaskList” on page 78
* “InsertIntoTasklist” on page 144
* “Activate” on page 72 for more information about the Activate node.

Table 4‑12 ConcatenateTaskList Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| task\_list\_var | Yes | Case-packet variable of object type holding a list of tasks to be executed. | None | Object |
| variable | Yes | Case-packet variable of object type holding a list of tasks to be appended to first list. | None | Object |

**Example: 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>

</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 216 for more information about the RESERVATIONS variable
* “ReleaseResource” on page 210 for more information about the RESERVATIONS variable

Table 4‑13 ConfirmResourceReservation Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| variable0, variable1... variableN | No | Indicates the variables that contain reserved resources. If no variable*N* 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 4‑14 CreateBean Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| Bean | Yes | Name of the JavaBean class that is used for creating the inventory bean object. | None | String |
| key\_field0 key\_field1 ... key\_fieldN | 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 |
| key\_value0 key\_value1 ... key\_valueN | Yes | Used in conjunction with the key\_field attributes to specify the values of the individual attributes in the JavaBean. | None | Any |
| bean\_variable | Yes | Name of the variable where the created JavaBean instance is returned | None | Object |

**Example: 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 4‑15 CreateInventory Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Name of the database module to be used. | “db” | String |
| bean | Yes | Name of the JavaBean class that is used for storing the data. | None | String |
| bean\_object | No | The name of the variable containing the inventory bean object to by stored in the inventory. | None | Object |
| key\_field0, key\_field1... key\_fieldN | 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 |
| key\_value0, key\_value1... key\_valueN | No | Used in conjunction with the key\_field attributes to specify the new value of the individual attributes in the JavaBean. | None | Any |
| bean\_variable | No | Name of the variable where the created/updated JavaBean instance is returned. | None | Object |
| strict\_create | 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 *strict\_update* parameter | “true” | Boolean |
| strict\_update | 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 *strict\_create* parameter. | “false” | Boolean |
| store\_audit | 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: 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.avtivator.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

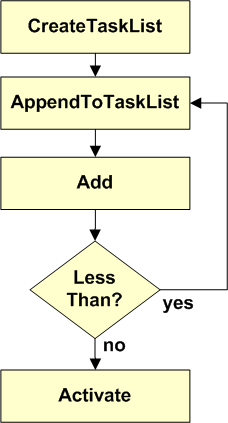
The node assigns a new task list to a case-packet variable of type Object. Tasks are then added to the list by the AppendToTaskList node. After the list is constructed, it is executed as a single transaction by the Activate node.

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 78 for more information about adding individual tasks
* “ConcatenateTaskLists” on page 94
* “InsertIntoTasklist” on page 144
* “Activate” on page 72 for more information about the Activate node

Table 4‑16 CreateTaskList Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| task\_list\_var | 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: 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>

</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.

### 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 |
| --- | --- | --- | --- | --- |
| in\_date\_millis | No | Milliseconds since January 1, 1970 00:00:00.000 GMT.  NOTE: cannot be used if in\_date\_string, in\_date\_seconds, or in\_current\_time is defined. | None | Numeric |
| in\_date\_seconds | No | Seconds since January 1, 1970  00:00:00.000 GMT.  NOTE: cannot be used if in\_date\_string, in\_date\_millis, or in\_current\_time is defined. | None | Numeric |
| in\_date\_string | No | Date formatted according to the date\_string\_format string.  NOTE: cannot be used if in\_current\_time or in\_date\_millis is defined. date\_string\_format must be defined. | None | Numeric |
| in\_current\_time | 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 in\_date\_string, in\_date\_seconds, or in\_date\_millis is defined. | None | String |
| return\_formatted\_ date | 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 |
| return\_date\_field | 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 |
| return\_date\_seconds | 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 |
| return\_date\_millis | 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 |
| result | Yes | String variable where returned result is to be stored. | None | String |
| date\_string\_format | No | The format of the in\_date\_string,  e.g. yyyyMMddhhmm.  NOTE: cannot be used without in\_date\_string. | None | String |

**Example: 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.

<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 node transforms an encrypted string to clear text.

Use this node just before the password must be used in clear text as it recommented 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 |
| encrypted\_text | Yes | The string to be decrypted. | None | String |
| decrypted\_text | Yes | The decrypted text output (clear text) | None | String |

**Example: 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 194

Table 4‑19 DeleteCache Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| caching\_module | Yes | Identified which caching module the object is saved in. | None | String |
| delete\_all | 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). | false | Boolean |
| bean | 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 |
| find\_by\_method | No | Then name of the method which was used when inserting the instance in the caching module. | findByPrimaryKey | String |
| key\_value0,  key\_value1  ... key\_valueN | 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 |
| db | No | Name of the database module to be used. | “db” | String |
| bean | Yes | Name of the JavaBean class that is used for deleting data. | None | String |
| key\_field0, key\_field1 ... key\_fieldN | 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 |
| key\_value0, key\_value1 ... key\_valueN | Yes | Used in conjunction with the key *key\_field* attributes to specify the key values of the data being deleted. | None | Any |
| store\_audit | 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: 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 385

Table 4‑21 DeleteScheduledJob Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| scheduled\_job\_id | 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 |
| db | No | Database module to use in order to perform the query. | “db” | String |
| service\_id | Yes | A case-packet variable that holds the unique identifier for the service instance that is being deleted. | None | Integer |

**Example: 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>

### 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 4‑23 DoNothing Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| Message | No | The message to be logged when the node is entered. A standard message is printed on exit. | None | String |

**Example: 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: 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 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 4‑24 Encrypt Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| Text | Yes | The string to be encrypted | None | String |
| encrypted\_text | Yes | The encrypted text output | None | String |

**Example: 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 33

Table 4‑25 ExecuteMacreo Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| workflow\_name | Yes | The name of the workflow to execute | None | String |
| input0, input1, ... input*N* | 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 |
| output0, output1, ... output*N* | 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: ExecuteMacro - use in the workflow**

The example below uses the ExecuteMacro node to invoke the workflow “macroChild”.

<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"/>

<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 4‑26 Equal Parameters

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

**Example: 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 node named “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 4‑27 ExecSQLQuery Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Specifies the database module to use in order to gain access to a database. | “db” | String |
| query | 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 |
| param0, param1 ... paramN | 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. 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 col<n> for the value. | None | Depends on the SQL query column type |

**Example: 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= ?"/>

<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 4‑28 ExecSQLStatement Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | This parameter specifies the database module to use in order to access a database. | “db” | String |
| statement | 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 |
| param0, param1... paramN | 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: 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.

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 4‑29 ExecuteExternal Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| cmd\_line | 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 *paramN* parameters. | None | String |
| wait | 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 |
| param0, param1... paramN | No | Indicates the names of case-packet variables whose values are used to replace the free variables (%s) in the cmd\_line. Specify as many param parameters as necessary (param0, param1...paramN). | None | Any |
| output\_var | No | Indicates the name of a variable to capture the output of the executed program. | None | Object |
| cwd | No | Indicates the working directory in which the command should be run. | *$ACTIVATOR\_VAR* | Object |
| variable0, variable1... variableN | No | Indicates the names of case-packet variables that should be passed to the running program on its stdin. If unspecified, each variable in the case-packet will be sent. | None | Any |

**Example: 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 4‑30 ForEach Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| list | 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 java.util.regex.Matcher, or a string where the elements are defined based on the patterns and groups defined in java.util.regex.Matcher. | None | String, Collection, Array, Object |
| element | Yes | Contains the last iterated element from the list. |  | Object |
| remaining | Yes | Contains the remaining elements in the list to be iterated. | None | Object |
| idx | No | Contains the index of the last iterated element. The index starts from 0. | None | Integer |
| count | No | Contains the number of iterations. The count starts from 1. | None | Integer |
| tag | No | Contains the XML String and the elements in the string list are iterated based on the defined XML String. | None | String |
| separator | No | Contains the separator pattern string and the elements in the list are iterated based on the defined pattern string.  Patterns are defined in  java.util.regex.Matcher. | None | String |
| skip\_if\_empty | 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 |
| pattern | 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 java.util.regex.Matcher | None | String |

**Example: 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="listSeperatorString" value="AB, C,"/>

<Variable-Value name="seperatorStr" 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 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 4‑31 GenericUIDialog Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| title | Yes | Title of the GenericUIDialog Interactable window | None | String |
| queue | Yes | Queue in which the request will wait | None | String |
| dialog\_data0 dialog\_data1... dialog\_dataN | 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 $ACTIVATOR\_VAR) containing failure details. The syntax is db:message\_id or file:file\_path. Cannot be specified if an identifier is configured | None | String |
| identifier | No | An identifier that points to activation dialogs in the database. Cannot be specified if dialog\_data is configured | None | String |
| dialog\_label | No | Label for the dialog details. Required only if dialog\_data or identifier is specified | None | String |
| 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 ... variable | 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 |
| 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 the 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: 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: 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"/>

<Varaible 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.

* Look for the section with the comment “Interact with running jobs (GenericUIDialog node”
* Set the value of the parameter customizeGenericUIDialogNodeNodeJSP to “true.”
* 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 4‑32 GetBaseFileName Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| file\_var | Yes | Variable containing the file name. The file name is placed in this variable unless *output\_var* is supplied. | None | String |
| output\_var | 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 Micro Focus NNMi\* 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 constrait\_value must also be specified.

Table 4‑33 GetBeansNNMNode Available Attributes

| Bean Type | Available Attributes |
| --- | --- |
| 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 |
| 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 4‑34 GetBeansNNMNode Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| module\_name | Yes | The name of the Micro Focus NNMi\* module used to connect to a specific Micro Focus NNMi\* server | None | String |
| bean\_type | 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 |
| result\_var | Yes | The case packet variable where the operation result will be stored | None | Object |
| condition\_name0, condition\_name1,  ....  condition\_nameN | False | Condition name for filtering purposes | None | String |
| condition\_value0, condition\_value1,  ....  condition\_valueN | False | Condition value for filtering purposes | None | String |
| condition\_operator0, condition\_operator1,  ....  condition\_operatorN | False | Condition value for filtering purposes. The following values can be specified:  {"EQ", "NE", "LT", "GT", "LE", "GE", "LIKE", "NOT\_IN"} | None | String |
| constraint\_name0, constraint\_name1,  ...  constraint\_nameN | 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 |
| constraint\_name0, constraint\_name1,  ...  constraint\_nameN | False |  | None | String |
| constraint\_value0, constraint\_value1,  ...  constraint\_valueN | False | The parameter indicates how all the specified conditions and constrains must be joined within the expression query sent to the Micro Focus NNMi\*.  This is a mandatory parameter when either conditions or constraints are specified.  There are only two options for the expression\_operator parameter:  {"AND", "OR"}. | None | String |

**Example: 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: 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 HPE 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 4‑35 GetBusinessHoursAfterDuration 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 either of type Integer or String. The actual business time after the given duration is populated in the case-packet variable. | None | String/ Integer |
| calendar\_name | Yes | The name of the calendar which needs to be used. | None | String |
| date\_value | 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 date\_format parameter. | None | String |
| duration | Yes | This parameter indicates the duration which will be added to the value of the date\_value parameter to get the business hour. The value of this parameter must be a positive integer. | None | Integer |
| time\_unit | No | The allowed values are "hours" or "minutes". | minutes | String |
| date\_format | 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' Locale's date format is used | String |
| timezone | No | The timezone in which the date\_value 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 HPE Service Activator is running. | String |

**Example: 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.

GetBusinessHoursAfterDuration

</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.

GetBusinessHoursAfterDuration

</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 4‑36 GetCalendarTimezone 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 String. | None | String/ Integer |
| calendar\_name | Yes | The name of the calendar which needs to be used. | None | String |

**Example: 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 HPE 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 4‑37 GetNextIncludedTime 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 thecase-packet variable should be either of type Integer or String. The next included time is populated in the case-packet variable. | None | String/ Integer |
| calendar\_name | Yes | The name of the calendar which needs to be used. | None | String |
| date\_value | 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 date\_format parameter | None | String |
| date\_format | 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' Locale's date format is used | String |
| timezone | No | The timezone in which the date\_value 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 HPE Service Activator is running. | String |

**Example: 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 HPE 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 4‑38 GetTimeRangesOfBusinessDay Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| start\_time\_range | 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 day\_of\_week parameter has been specified then the casepacket variable type must be string. | None | String/ Integer |
| end\_time\_range | 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 day\_of\_week parameter has been specified then the casepacket variable type must be string. | None | String/ Integer |
| calendar\_name | Yes | The name of the calendar which needs to be used. | None | String |
| date\_value | 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 |
| date\_format | 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: 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="calender\_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 4‑39 GetOperatingSystem Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| output\_var | Yes | The name of the case packet variable to return the operating system type. | None | String |
| throw\_excep | 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: 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 4‑40 GreaterThan Parameters

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

**Example: 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>

### GreaterThanOrEqual

com.hp.ov.activator.mwfm.component.builtin.GreaterThanOrEqual

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 4‑41 GreaterThanOrEqual Parameters

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

**Example: GreaterThanOrEqual - 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.GreaterThanOrEqual

</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 that the worker thread which is used to execute the node is *not* freed while sending the HTTP get request and waiting for the response.

Table 4‑42 HTTPGet Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| URL | Yes | The target URL for the HTTP(S) connection. | None | String |
| username | No | Username for network connection authentication | None | String |
| password | No | Password for network connection authentication | None | String |
| keystore | No | The location of the keystore file, necessary for HTTPS client authentication. | None | String |
| storepass | No | The password to access the keystore file, necessary for HTTPS client authentication. | None | String |
| keypass | No | The password for the public certificate/private key pair, necessary for HTTPS client authentication.  Note that the parameters *keystore*, *storepass*, and *keypass* 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 |
| proxy\_server | No | Name of a proxy server. | None | String |
| proxy\_port | No | Port of a proxy server.  Note that the parameters proxy\_server and proxy\_port 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 |
| cookie | No | Cookie of the HTTP(S) request | None | String |
| connect\_timeout | No | Connection timeout value, in milliseconds. | None | String |
| read\_timeout | No | Read timeout value, in milliseconds. | None | String |
| Response | Yes | Case-packet variable holding the HTTP(S) response. | None | String |
| return\_cookie | No | Case-packet variable holding the returned cookie, if any. | None | String |

**Example: 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 4‑43 HTTPRequest Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| request\_type | Yes | The type of request. It can either POST or GET | None | String |
| content\_type | No | Content-Type header to be sent in the request. | text/xml | String |
| module | Yes | The name of the HTTP module | None | String |
| request | 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 |
| return\_co okie | No | Returned cookie value | None | String |
| response | Yes | Returned result for the HTTP(S) GET or POST request | None | String |

**Example: 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: 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:HPE 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 99 for more information about creating a new task list.
* “ConcatenateTaskLists” on page 94
* “Activate” on page 72 for more information about the Activate node.
* “AppendToTaskList” on page 78 for more information about appending a task to the end of a task list.

Table 4‑44 InsertIntoTasklist Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| task\_list\_var | Yes | Indicates the variable containing the task list to insert into. (Created using CreateTaskList) | None | Object |
| task | Yes | A variable or a constant containing the name of the task to be inserted in the list. | None | String |
| param0, param1,..., paramN | 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 |
| position | No | Position of the new task in the list after insert is performed. | 0 | Integer |

**Example: 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>

### 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 4‑45 InvokeInventoryMethod Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| Db | No | Name of the database module to be used. | “db” | String |
| Bean | Yes | Name of the JavaBean to be used. | None | String |
| arg0 arg1... argN | Yes | Arguments to be passed to the method. If its value begins with constant: 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 |
| variable | Yes | Case-packet variable holding the invoked method’s return value. | None | String |
| method | No | Name of the method to be invoked. | “findBy Primary Key” | 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 parameters.

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 4‑46 InvokeMethod – Conversion to Boolean Type

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |
| 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 4‑47 InvokeMethod Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Name of the database module to be used if the method takes a Connection argument. | “db” | String |
| bean | 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 |
| arg0 arg1... argN | 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 java.sql.Connection, it is passed implicitly, and should not be listed among the arg parameters to InvokeMethod | None | Any |
| variable | Yes | Case-packet variable holding the invoked method’s return value. | None | Any |
| method | Yes | Name of the method to be invoked. | None | String |
| persist | No | List of comma-separated input variable names that should be marked as updated for persistence. Write bean if the bean is an object. Write \* if all input variables should be persisted. | None | String |

### IsModuleConfigured

com.hp.ov.activator.mwfm.component.builtin.IsModuleConfigured

The node checks if a specified Workflow Manager module is configured.

Table 4‑48 IsModuleConfigured Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| module | 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 4‑49 IsTimeIncluded Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| response | Yes | The name of the case-packet variable name in which the result is stored. | None | String/ Integer |
| calendar\_name | Yes | The name of the calendar which needs to be used. | None | String |
| date\_value | 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 date\_format parameter. | None | String |
| date\_format | 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' Locale's date format is used | String |
| timezone | No | The timezone in which the date\_value 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 HPE Service Activator is running. | String |

**Example: 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 4‑50 IsTrue Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| op1 | Yes | The name of a Boolean case-packet variable, whose value must be tested. | None | Boolean |

### JavaToJson

com.hp.ov.activator.mwfm.component.builtin.JavaToJson

The node will produce the json representation of a Java Object.

Table 4‑51 JavaToJson Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| input\_var | Yes | Name of the case packet variable that stores the Java Object whose json represention is going to be obtained. | None | Object |
| pretty\_print | No | If *true* then the resulting json will contain indentation and line break characters. | false | Boolean |
| variable | Yes | Name of case packet variable where the json representation of the input object will be stored. | None | String |

**Example: JavaToJson - use in a workflow**

<Process-Node>

<Name>Java To Json</Name>

<Description>Produce json</Description>

<Action>

<Class-Name>

com.hp.ov.activator.mwfm.component.builtin.JavaToJson

</Class-Name>

<Param name="input\_var" value="bean"/>

<Param name="variable" value="json"/>

</Action>

</Process-Node>

...

<Case-Packet>

<Variable name="bean" type="Object"/>

<Variable name="json" type="String"/>

</Case-Packet>

### JsonToJava

com.hp.ov.activator.mwfm.component.builtin.JsonToJava

The node will instanciate a Java Object from json text string.

Table 4‑52 JsonToJava Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| input\_json\_var | Yes | JSON text to be instanciate | None | String |
| input\_json\_rootname\_var | No | JSON Root Name - needed if class name does not match the jsonRoot. The value NONE indicates no root name. | “NONE” | String |
| output\_class | Yes | The class name of the Object to be created. | None | String |
| output\_var | Yes | Name of case packet variable to catch the Object that was created. | None | Object |

### 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 4‑53.

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 HPE 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 HPE 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 clas s 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 HPE 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 next.

Table 4‑53 Java Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| javafile | 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 |
| javacode | No | The contents of a java template as defined by <jtp> format.  Note that it should only be used for minor operations. | None | String |
| method | 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 |
| expression | No | Simple Java expression. | None | String |
| arg0...argN | 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 argume- nt type |
| output\_var | No | Optional variable for the return value. Type of the variable should coincide with return type of the executed method. | None | Object |
| cleanup | No | Optional string containing the name of a method to be executed in the NodeExited method. See javadoc Class WFProcessNode. | None | String |
| strict\_scope | No | This Java node adheres to scoping rules(As explained above) | True | Boolean |
| in\_scope | No | List of variables brought into scope if strict\_scope is true. | None | String |
| use\_solution\_dir | No | When set to "true", the nodes will read from $SOLUTION\_ETC/template\_fi les instead of $ACTIVATOR\_ETC/template\_files. | false | Boolean |

**Example: 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-Name>

<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: Java Template (jtp) File**

import com.fut.byt.\* 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 below.

**Example: 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.byt.\*;

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 this example. 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 4‑54.

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 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 WFException, and workflow execution will fail

**See Also**

* “Java” on page 154 for more information about the parameters and Java code usage

Table 4‑54 JavaRule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| javafile | 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 |
| javacode | No | Java functional declaration. | None | String |
| method | No | Name of the method to call. | None | String |
| expression | No | Java expression. Expression can be combined with javafile/javacode and can be wrapped in an extra dummy method. | None | String |
| arg0...argN | No | Parameters or arguments to pass to the method; cannot be combined with expression. | None | Depends on the method argument type |
| strict\_scope | No | This Java node adheres to scoping rules(As explained above) | True | Boolean |
| in\_scope |  | List of variables brought into scope if strict\_scope is true. | None | String |
| use\_solution\_dir | No | When set to "true", the nodes will read from $SOLUTION\_ETC/template\_fi les instead of $ACTIVATOR\_ETC/template\_ files. | false | Boolean |

**Example: 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 154 for more information about the parameters and Java code usage

Table 4‑55 JavaSwitch Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| javafile | 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 |
| javacode | No | Java functional declaration. | None | String |
| method | No | Name of the method to call. | None | String |
| expression | No | Java expression. Expression can be combined with javafile/javacode and can be wrapped in an extra dummy method. | None | String |
| arg0...argN | No | Parameters or arguments to pass to the method; cannot be combined with expression. | None | Depends on the method argument type |
| strict\_scope | No | This Java node adheres to scoping rules (As explained above) | True | Boolean |
| in\_scope |  | List of variables brought into scope if strict\_scope is true. | None | String |
| computed\_key | No | Optional variable to capture the computed value of switch key | None | Object |
| case0...caseN | 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. |
| 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 |

**Example: 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 4‑56 KillJob Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| job\_id | 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: 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 4‑57 LessThan Parameters

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

**Example: 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 4‑58 LessThanOrEqual Parameters

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

**Example: 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 4‑59 Log Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| component\_name | Yes | Name of the component logging the message. | None | String |
| service\_id | No | This parameter is used during Solution Logging. The default value is the system case-packet SERVICE\_ID. | SERVICE\_ID | String |
| log\_level | Yes | Indicates Logging levels. The value should be DEBUG, DEBUG2, INFORMATIVE,WARNING, or ERROR. | None | String |
| log\_message | Yes | The message to be logged. | None | String |
| param0,  param1,  ... paramN | No | Parameters to replace free variables in the message. | None | String |
| part\_name | No | Indicates the name of the part logging the message. The value can be either FRAMEWORK or COMPONENT. | COMPONENT | String |
| topic\_name | No | Indicates the name of the topic for the message. The value can be NO\_TOPIC, TOPIC\_STATISTICS, TOPIC\_STARTUP, TOPIC\_RECOVERY, TOPIC\_COMMON\_OPERATION, or TOPIC\_SHUTDOWN. | NO\_TOPIC | String |
| log\_manager | No | Name of the log module used for storing and accessing the logs. | log\_manager | String |

**Example: 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>

### MailRequest

com.hp.ov.activator.mwfm.component.builtin.notification.mail.MailRequest

This node uses the MailRequestModule in order to send emails in a synchronous way. In this context, *synchronous* means that the workflow node execution will keep waiting until it gets the confirmation of the mail reception from the email server.

**NOTE** The email could be rejected afterwards due to several reasons, such as: wrong recipient, exceeding size limitation, and so on.

It is required to configure the module beforehand, by specifying the connection parameters, such as the email server hostname, the email server port, and so on.For more details, see *Section 6.23 MailRequestModule*.

**NOTE** Itis mandatory to specify at least one of the following input parameters: *to*, *to*\_*list*, *bcc*, *bcc*\_*list*, *cc,* or *cc\_list*.

Table 4‑60 MailRequest Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| module\_name | Yes | The name of the associated MailRequestModule | None | String |
| body | Yes | The content of the message | None | String |
| subject | Yes | The subject of the message | None | String |
| to0 to1 … toX | No | The email address of the email’s recipient. | None | String |
| to\_list | No | A list of addresses of the email’s recipient. | None | List |
| cc0 cc1 … ccX | No | The carbon copy email address. | None | String |
| cc\_list | No | A list of carbon copies. | None | List |
| bcc0 bcc1 … bccX | No | The blind carbon copy email address. | None | String |
| bcc\_list | No | A list of blind carbon copies. | None | List |
| from | No | The email address of the sender | None | String |
| mail\_format | No | It defines the format of the content: TEXT or HTML | TEXT | String |
| attachment0 attachment1 … attachmentN | No | A list of filenames to be attached to the message. The files must be located in the folder: *$ACTIVATOR\_VAR*/mail | None | List |
| priority | No | It supports three levels: 5 – Low, 3 – Normal, 1 – High . The value will be included in the mail header “X-Priority” | None | Number |

**Example: MailRequest - use in the workflow**

This example assumes that the target variable contains a text with the email’s recipient and the module is also well configured.

<Process-Node disablePersistence="true">

<Name>SendMail</Name>

<Action>

<Class-Name>com.hp.ov.activator.mwfm.component.builtin.notification.mail.MailRequest</Class-Name>

<Param name="body" value="constant:We&apos;ll release a new version very soon."/>

<Param name="from" value="constant:team.hpsa@hpe.com"/>

<Param name="module\_name" value="constant:mail\_request\_module"/>

<Param name="subject" value="constant:New release is coming ..."/>

<Param name="to0" value="target"/>

</Action>

</Process-Node>

</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](#_bookmark30)). 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 4‑61 MapData Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| map\_var | Yes | Case-packet variable of type Object that contains the map. | None | Object |
| name of a case-packet variable | Yes | The name provided is the name of a case-packet variable. The value provided is a key into the map indicated by *map\_var*. The call to MapData 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: 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** The example 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 4‑62 MatchDBQuery Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Specifies the database module to use in order to perform the query. | “db” | String |
| query | 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 |
| attribute\_name | No | The name of the column to be used as the attribute name. | “name” | String |
| attribute\_value | No | The name of the column to be used as the attribute value. | “value” | String |
| param0, param1... paramN | Yes | A series of these parameters can be used to specify the values for free variables (question marks) in the statement. | None | String |
| no\_error | No | If a row is selected but it is not in the case-packet then no exception is thrown if *no\_error* is set to true. | “false” | Boolean |

**Example: 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: 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 4‑63 MatchDBStore Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Specifies the database module to use in order to perform the statement. | “db” | String |
| statement | Yes | SQL statement to be carried out for each variable. | None | String |
| attr\_name\_col | 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 |
| attr\_value\_col | 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 |
| param0, param1... paramN | No | A series of these parameters can be used to specify the values for free variables (question marks) in the statement. | None | String |
| variable0, variable1... variableN | Yes | List of case-packet variables to be stored. There can be as many variable parameters as needed. | None | String |

**Example: 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"/>

<Action>

...

<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"

|  |  |  |
| --- | --- | --- |
| Name | Value | Customer Id |
| 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 InvokeMetod 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\_excep argument is false, then the rule will follow the false-arrow of the rule in the workflow.

Table 4‑64 MethodRule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Name of the database module to be used if the method takes a Connection argument. | “db” | String |
| bean | 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 |
| arg0 arg1... argN | 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 java.sql.Connection, it is passed implicitly, and should not be listed among the arg parameters to InvokeMethod | None | Any |
| variable | Yes | Case-packet variable holding the invoked method’s return value. | None | Any |
| method | Yes | Name of the method to be invoked. | None | String |
| persist | No | List of comma-separated input variable names that should be marked as updated for persistence. Write bean if the bean is an object. Write \* if all input variables should be persisted. | None | String |

### MethodSwitch

com.hp.ov.activator.mwfm.component.builtin.MethodSwitch

The MethodSwitch node works in the same way as the InvokeMetod node. The difference is that MethodSwitch is a switch node, which will branch according to the result of the invoked method. The result is interpreted as a case according to the switch defined for the return-value of the InvokeMethod. If the method throws an exception, and the throw\_excep argument is false, then the switch node will follow the default branch in the workflow.

Table 4‑65 MethodSwitch Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Name of the database module to be used if the method takes a Connection argument. | “db” | String |
| bean | 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 |
| arg0 arg1... argN | 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 java.sql.Connection, it is passed implicitly, and should not be listed among the arg parameters to InvokeMethod | None | Any |
| case0 case1... caseN | No | Case values specified for various paths | None | Any |
| method | Yes | Name of the method to be invoked. | None | String |
| persist | No | List of comma-separated input variable names that should be marked as updated for persistence. Write bean if the bean is an object. Write \* if all input variables should be persisted. | 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 221.

Table 4‑66 ModifyScheduleJob Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| scheduled\_job\_id | Yes | The ID of the scheduled job you want to modify. | None | Integer |
| schedule\_time | 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 |
| group\_id | 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 |
| reoccurence\_freq | No | Reoccurrence frequency period. This parameter has to be specified if the scheduled job has to be run repeatedly. If reoccurence\_freq is not specified, the parameter accepts the value as seconds. The value should be numeric. | 0 | Integer |
| reoccurence\_end\_time | No | The time when reoccurrence of schedule job must end. This parameter accepts the date and time as millisceonds with the value starting value from January 1, 1970 00:00.00:000 GMT. The value should be numeric. | 0 | Integer |
| description | No | Description of a scheduled job. The default value is an empty string. | None | String |
| status | No | Status of a scheduled job. The default value is an empty string. | None | String |
| reoccurrence\_freq\_units | 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 4‑67 MoveFile Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| file | 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 *$ACTIVATOR\_VAR* | None | String |
| destination | Yes | The destination of the moved file. You can specify a case-packet variable or a constant. This can be a new file name or a directory. If the path is relative, it will be interpreted as relative to *$ACTIVATOR\_VAR* | 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 259
* “Assign” on page 86

Table 4‑68 MultiAssign Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| variable0 variable1  ... variableN | Yes | Case-packet variables to be set. | None | String / Integer / Float / Boolean / Object |
| value0 value1 ... valueN | Yes | 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. |
| default0 default1... defaultN | No | Default value to be used if the corresponding value is null, empty, or 0. | None | String / Integer / Float / Boolean / Object |

**Example: 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 4‑69 Multiply Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| op0 | 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 |
| op1 | Yes | Name of a case-packet variable that is the second variable to be multiplied. A hard-coded value can be specified using constant:*X* where *X* is the constant value. | None | Numeric |

**Example: 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 182
* "MutexUnlock” on page 185
* "MutexSetInfo” on page 184
* "ConflictModule” on page 314

A text string the job wants to be possible for other jobs to read while the job has locked the mutex.

Table 4‑70 MutexGetInfo Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| solution | No | The solution name | None | String |
| key | Yes | The key name | None | String |
| value | Yes | The value for the key | None | String |
| info\_string | 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 accross 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 181.
* “ConflictModule” on page 314.

Table 4‑71 MutexLock Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| solution | No | The solution name | None | String |
| key | Yes | The key name | None | String |
| value | Yes | The value for the key | None | String |
| info\_string | No | A text string the job wants to be possible for other jobs to read while the job has locked the mutex. | None | String |
| wait | No | Indicating if the node should wait until mutex is freed if already locked by another job | None | Boolean |
| failed\_when\_deadlock | No | Indicating if the node should failed if a deadlock is found. If set to false then some of the previous locked mutexes would have been freed and used by other jobs before they are locked by this job again. | true | Boolean |
| result | Yes, if wait is set to false | The Case-packet which should contain the result of if the mutext 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 |
| locked\_job\_id | No | The id of the job currently holding the lock. | None | Integer |
| locked\_info\_string | No | The info string value of the job currently holding the lock. | None | String |

**Example: 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 185.
* “MutexGetInfo” on page 181.
* “ConflictModule” on page 314.

Table 4‑72 MutexSetInfo Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| solution | No | The solution name | None | String |
| key | Yes | The key name | None | String |
| value | Yes | The value for the key | None | String |
| info\_string | 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 fails.

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 182.
* “ConflictModule” on page 314.

Table 4‑73 MutexUnlock Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| solution | No | The solution name | None | String |
| key | Yes | The key name | None | String |
| value | Yes | The value for the key | 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 4‑74 Not Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| op1 | 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: Not - use in the workflow**

This example verifies whether the val ue 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 HPE 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**

* “PPU” on page 190

Table 4‑75 PAYG Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| unit | Yes | A string indicating the name of a unit in a solution which needs to be licensed | None | String |
| solution | No | A string indicating the name of a solution which needs to be licensed | None | String |
| increment | No | The increment count value which needs to be added to the Used count. | 1 | Integer |

**Example: 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>

### ParseTemplate

com.hp.ov.activator.mwfm.component.builtin.ParseTemplate

The node will parse a Velocity template and substitute embedded variables. The input parameter template must contain a velocity template string and the template\_output will contain the parsed template string.

The parameters key/value and template\_map are mutual exclusive.

Table 4‑76 ParseTemplate Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| template | Yes | Template to parse | None | String |
| template\_map | Yes – if no key parameter is defined | The map holding the variables to be substituted | None | Object |
| template\_output | Yes | The template after being parsed. | None | String |
| key0, key1, ... keyN | Yes if template\_map is not defined | The keys to be populated in the velocity context. "key#" will be mapped with "value# | None | String |
| Value0, value1, ... | Yes if key is provided | The value of the key to be populated in the velocity context. "key#" will be mapped with "value#" | None | Object |

**Example ParseTemplate - use in the workflow**

This example provides a set of key/values to be populated in the string case-packet template:.

<Process-Node disablePersistence="true">

<Name>ParseTemplate</Name>

<Description></Description>

<State></State>

<Action>

<Class-Name>com.hp.ov.activator.mwfm.component.builtin.ParseTemplate</Class-Name>

<Param name="output\_var" value="result"/>

<Param name="template" value="template"/>

<Param name="key0" value="constant:SERVER\_NAME"/>

<Param name="value0" value="constant:test\_server\_name"/>

<Param name="key1" value="constant:AVAILABILITY\_ZONE"/>

<Param name="value1" value="constant:"/>

<Param name="key2" value="constant:FLAVOR"/>

<Param name="value2" value="constant:gold"/>

<Param name="key3" value="constant:IMAGE"/>

<Param name="value3" value="constant:vsr\_image"/>

<Param name="key4" value="constant:AVAILABILITY\_ZONE"/>

<Param name="value4" value="constant:availability\_zone\_name"/>

<Param name="key5" value="constant:HOST"/>

<Param name="value5" value="constant:host\_name"/>

</Action>

<Next-Node>PutMessage</Next-Node>

</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 4‑77 PatternMatch Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| Value | Yes | The string to be processed | None | String |
| Pattern | Yes | The regular expression as defined by the java.util.regex.Matcher class. | None | String |
| Startat | No | Variable indicating the index where matching starts. Updated to index after each match. | 0 | Integer |
| multiline | No | Pattern spans multiple lines | False | Boolean |
| Nocase | No | Matching case is insensitive | False | Boolean |
| Match | No | Result of the entire match | None | String |
| Group | 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 HPE 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**

* “PAYG” on page 187.

Table 4‑78 PPU Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| unit | Yes | A string indicating the name of a unit in a solution which needs to be licensed | None | String |
| solution | No | A string indicating the name of a solution which needs to be licensed | None | String |
| increment | No | The increment count value which needs to be added to the Used count. | 1 | Integer |

**Example: 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 4‑79 PutMessage Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| queue | 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 |
| message | 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 |
| param0, param1, ... param*N* | 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 |
| service\_id | No | The Service Identifier value used for associating the message with a solution. | None | String |
| order\_id | No | The Order Identifier value used for associating the message with a solution. | None | String |
| Type | No | The type value of the workflow. | None | String |
| State | No | The state value of the workflow. | None | String |
| Skip | No | If true then the message will not be written | false | Boolean |

**Example: 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"/>

<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: 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="operand1"/>

<Param name="queue" value="variable:myqueuevar"/>

</Action>

</Process-Node>

<Case-Packet>

<Variable name="operand1" type="Integer"/>

</Case-Packet>

**Example: 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="operand1"/>

<Param name="queue" value="sum\_queue"/>

<Param name="service\_id" value="serviceId1"/>

</Action>

</Process-Node>

<Case-Packet>

<Variable name="operand1" type="Integer"/>

</Case-Packet>

**Example: 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="operand1"/>

<Param name="queue" value="sum\_queue"/>

<Param name="service\_id" value="variable:SERVICE\_ID"/>

</Action>

</Process-Node>

<Case-Packet>

<Variable name="operand1" type="Integer"/>

</Case-Packet>

<Initial-Case-Packet>

<Variable-Value name="SERVICE\_ID" value="srvc1"/>

</Initial-Case-Packet>

**Example: 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 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 Chapter 2.

Table 4‑80 QueryInventory Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Name of the database module to be used. | “db” | String |
| bean | Yes | Name of the JavaBean to be used. | None | String |
| use\_cache | No | If set to true then the result will be saved in the configured caching module. | false | Boolean |
| caching\_module | Yes if use\_cache is set to true | The name of the caching module to use. | None | String |
| key\_value0, key\_value1... key\_valueN | Yes | Value of the key by which the element should be looked up. If its value begins with constant: the key is the value provided after this. Otherwise, the key indicates the name of a variable that holds the key value. | None | Object |
| variable | Yes | Case-packet variable holding the returned bean or bean array. | None | Object |
| find\_by\_method | No | Name of the method to use for query. | “findByPr imaryKey” | Integer |
| where | No | This parameter appends a simple SQL “*WHERE*” clause to “*SELECT*” statement. The parameter may not be used if the *find\_by\_method* parameter is set to “findByPrimaryKey”. Any %s symbols appearing in the string are replaced by consecutive paramN parameters. | None | String |
| preserve | 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 |
| param0, param1... paramN | 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 |
| return\_array | 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 *find\_by\_method* parameter | False | Object |

**Example: 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).

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 l ist 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 4‑81 QueryScheduledJob Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| scheduled\_job\_id | No | The ID of the scheduled job for which a SchedulerJobDescriptor must be returned. | 0 | Integer |
| ret\_result | 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 |
| where | 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 |
| param0, param1, ... paramN | 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 where parameter, param1 is replaced with the second and so forth. | None | String |
| order\_by | No | This parameter passes a simple SQL “ORDER BY” statement. | None | String |

**Example: 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>

</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>

</Process-Node>

**Example: 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>

</Process-Node>

### QueryServiceInstance

com.hp.ov.activator.mwfm.component.builtin.QueryServiceInstance

The node retrieves the values of service-instance parameters. The n ode 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 possi ble. 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 4‑82 QueryServiceInstance Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Database module to use in order to perform the query. | “db” | String |
| service\_id | 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 |
| variable0, variable1... variableN | No | Case-packet variables whose values are to be fetched. You can specify as many variable parameters as necessary. If you do not specify any variable parameters, all service instance parameters are fetched and the node expects case-packet variables to match each service-instance parameter. | None | String |

**Example: 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 4‑83 QueryServiceInstanceAll Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| service\_id | Yes | The ID of the service instance to fetch. | None | String |
| db | No | Name of the module to access the database. | “db” | 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 4‑84 RandomInteger Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| max\_number | No | Maximum random number generated - default is 100. | 100 | Integer |
| output\_var | Yes | Name of the case-packet variable to return the generated number. | None | Integer |

**Example: 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: 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 t he 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 usefull 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 cat 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 4‑85 ReadData Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| message\_url | No | Name of the case packet variable containing the message id. The syntax is db:<message\_id>, file:<file name>, or data:<string>. | None | String |
| job\_id | No | Name of the case packet variable containing the job id | None | String |
| module\_name | No | Name of the case packet variable containing the module name | None | String |
| Hostname | No | Name of the case packet variable containing the hostname | None | String |
| Identifier | No | Name of the case packet variable containing the identifier | None | String |
| output\_value | 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 |
| data\_length | No | The value of this parameter is the number of consecutive bytes of the data to be retrieved. | None | Integer |
| data\_position | No | The value of paramenter is the ordinal position from where the data byte has to be extracted. | None | Integer |
| return\_object\_type | 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 |
| Charset | No | The charset to be used when converting retrieved data to a String. | None | String |

**Example: 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: Retrieve Partial Data**

The case pa cket messageId contains the message id. The retrieved data would be stored in the case packet retrievedData, and it contains data up to ”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 cat 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 4‑86 ReadDataFromDatabase Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| message\_url | No | Name of the case packet variable containing the message id. The syntax is db:<message\_id>. | None | String |
| job\_id | No | Name of the case packet variable containing the job id | None | String |
| module\_name | No | Name of the case packet variable containing the module name | None | String |
| hostname | No | Name of the case packet variable containing the hostname | None | String |
| identifier | No | Name of the case packet variable containing the identifier | None | String |
| output\_value | 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 |
| data\_length | No | The value of this parameter is the number of consecutive bytes of the data to be retrieved. | None | Integer |
| data\_position | No | The value of paramenter is the ordinal position from where the data byte has to be extracted. | None | Integer |
| return\_object\_type | 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 |
| charset | No | The charset to be used when converting retrieved data to a String. | None | String |

**Example: 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: Retrieve Partial Data**

The case pa cket 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 4‑87 ReadFile Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| file | 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 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 read relative to *$ACTIVATOR\_VAR*. | None | String |
| destination | Yes | Name of the case-packet variable to put the contents of the file into. | None | String |
| charset | No | The charset to be used when converting retrieved data to a String. | None | String |

**Example: 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>

### ReceiveMessage

com.hp.ov.activator.mwfm.component.builtin.ReceiveMessage

Receives a message to start a new asynchronous transaction, which is stored as a service order registry and processed as such.

Table 4‑88 ReceiveMessage Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| workflow\_name | Yes | The name of the workflow to start. | None | String |
| transaction\_id | Yes | The transaction identifier. | None | String |
| system\_name | Yes | The name of the system from where this transaction is requested. | None | String |
| variable0,  variable1...  variableN | No | Case-packet variables that are to be passed to initialize variables in the new workflow being started. | None | Object |
| async\_descriptor\_variable | No | A case-packet variable to catch the asynchronous descriptor of the newly started transaction. | None | Object |
| destination0, destination1... destinationN | No | The name of a case-packet variable to initialize in the new workflow. By default the variable of the same name as the "variable" parameter is initialized. | None | Object |
| service\_id | No | The service identifier. | None | String |
| order\_id | No | The order identifier. | None | String |
| message | No | The message itself. | None | String |
| response\_config | No | The response configuration. | None | String |
| response\_config\_display | No | The public part of the response configuration to be eventually displayed in the UI. | None | String |
| parent\_transaction\_id | No | The parent transaction identifier. | None | String |
| throw\_excep | No | Controls whether the node should throw exceptions upon failures, or if the framework should handle them. If set to 'false' the framework will handle the failure by setting the RET\_VALUE case packet variable to -1. The RET\_TEXT variable will hold the failure text. | true | Boolean |

### ReceiveMessageAndWait

com.hp.ov.activator.mwfm.component.builtin. ReceiveMessageAndWait

Receives a message to start a new synchronous transaction, which is stored as a service order registry and processed as such, waiting for the transaction to finish.

Table 4‑89 ReceiveMessageAndWait Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| workflow\_name | Yes | The name of the workflow to start. | None | String |
| transaction\_id | Yes | The transaction identifier. | None | String |
| system\_name | Yes | The name of the system from where this transaction is requested. | None | String |
| variable0,  variable1...  variableN | No | Case-packet variables that are to be passed to initialize variables in the new workflow being started. | None | Object |
| destination0, destination1... destinationN | No | The name of a case-packet variable to initialize in the new workflow. By default the variable of the same name as the "variable" parameter is initialized. | None | Object |
| service\_id | No | The service identifier. | None | String |
| message | No | The message itself. | None | String |
| WORKFLOW\_ORDER\_ID | No | The order\_id to be passed to the child workflow. The child workflow will receive the order\_id value in the initial case packet variable named "WORKFLOW\_ORDER\_ID". | None | String |
| outputvar0,  outputvar1...  outputvarN | No | Case-packet variables that the child should pass back. | None | Object |
| queue | No | The name of the queue where the job will wait, and where the child job must do the synchronization (constant by default). Default queue is 'sync'. | “sync” | String |
| swap | No | Instructs the Workflow manager to swap-out the case-packets while the parent job waits for the child job to synchronize, in order to reduce memory footprint | None | Boolean |
| user\_name | Yes | The SOR user name. | None | String |

### RediscoverHostsNNMNode

com.hp.ov.activator.mwfm.component.builtin.nnmrequest.RediscoverHostsNNMNode

The node requests Micro Focus NNMi\* to rediscver 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 Micro Focus NNMi\* on a regular basic 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 Micro Focus NNMi\* system at that moment instead of relying on the automatic rediscover period.

Table 4‑90 RediscoverHostsNNMNode Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| module\_name | Yes | The name of the Micro Focus NNMi\* module used to connect to a specific Micro Focus NNMi\* server | None | String |
| host\_to\_rediscover0, host\_to\_rediscover1,  ... host\_to\_re discoverN | 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 216 for more information about the RESERVATIONS variable
* “ConfirmResourceReservation” on page 95 for more information about the RESERVATIONS variable

Table 4‑91 ReleaseResource Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Database module to use in order to perform the query. | “db” | String |
| variable0, variable1... variableN | 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 *RESERVATIONS* variable are released. | None | Object |

**Example: 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.

Alternativly 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 4‑92 RemoveData Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| url\_name | No | Name of the case-packet variable holding the message id or the file path ( an absolute path, or a filename relative to *$ACTIVATOR\_VAR*) to be removed. The syntax is db:message\_id or file:file\_path. | None | String |
| job\_id | No | Name of the case-packet variable holding the job id . | None | String |
| identifier | No | Name of the case-packet variable holding the identifier. | None | String |
| module\_name | No | Name of the case-packet variable holding the module\_name. | None | String |
| hostname | No | Name of the case-packet variable holding the hostname. | None | String |
| delete\_count | No | This case packet variable, if defined, will contain the count of rows deleted by the node. | None | Integer |

**Example: 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: 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: 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 4‑93 RemoveFile Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| File | 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 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 read relative to *$ACTIVATOR\_VAR*. | None | String |

### Replace

com.hp.ov.activator.mwfm.component.builtin.Replace

The node carries out simple character substitutions.

Table 4‑94 Replace Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| variable | Yes | Name of the case-packet variable in which substitutions are performed. | None | String |
| origin | Yes | Character to be replaced. You can specify \nto indicate a carriage return. | None | String |
| destination | Yes | Destination character. You can specify \n. | None | String |

**Example: 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 reser ved 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 resourc es 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 the be used to ensure that all reserved resources are released (for instance, in case of unexpected errors)

**See Also**

* “ReleaseResource” on page 210 for more information about the RESERVATIONS
* “AppendToResourceList” on page 77
* “ConfirmResourceReservation” on page 95 for more information about the RESERVATIONS

Table 4‑95 ReserveResource Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| db | No | Database module to use in order to perform the query. | “db” | String |
| bean0, bean1... beanN | 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 |
| key\_field0, key\_field1... key\_fieldN | No | Name of the field by which the reservation is carried out. This defaults to doing a reservation by PrimaryKey. | None | String |
| key\_value0, key\_value1... key\_valueN | 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 |
| variable0, variable1... variableN | Yes | Name of a case-packet variable in which to catch the reserved resource. | None | Object |
| resource\_list\_var | 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 |
| output\_list\_var | 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: 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: 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 4‑96 RetrieveSequence Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | The db module used to connect to the database. If unspecified in the workflow, it defaults to ‘db’. | “db” | String |
| sequence\_name | Yes | Name of the specific sequence whose next value has to be retrieved from the database. | None | String |
| sequence\_value | No | Holds the next value of the sequence as retrieved from the database by the process node. | None | Object |

**Example: 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 haveing 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 4‑97 ScheduleCurrentJob Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| schedule\_time | 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 |
| sleep\_time | No | Sleep time to supend the running job for a certain period of time. The sleep time must be specified in seconds. | 0 | Integer |
| Group | No | The group id of scheduled job. | 0 | String |
| description | No | The descrioption of scheduled job. | 0 | String |
| Status | 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.

Table 4‑98 ScheduleJob Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| *workflow\_name* | Yes | The name of the workflow to schedule | None | String |
| *schedule\_time* | 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 |
| *group\_id* | 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 |
| *reoccurrence\_freq* | No | Reoccurrence frequency period. This parameter has to be specified if the scheduled job has to be run repeatedly. If reoccurence\_freq is not specified, the parameter accepts the value as seconds. The value should be numeric. | 0 | Integer |
| *reoccurrence\_end\_time* | No | The time when reoccurrence of schedule job must end. This parameter accepts the date and time as millisceonds with the value starting value from January 1, 1970 00:00.00:000 GMT. The value should be numeric. | 0 | Integer |
| *description* | No | Description of the scheduled job. The default value is an empty string. | None | String |
| *status* | No | Status of the scheduled job. The default value is an empty string. | None | String |
| *variable0, variable1,... VariableN* | 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 |
| *destination0, destination1,.. destinationN* | 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 |
| *ret\_scheduled\_job\_id* | No | If the scheduled job was succesfully 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 |
| *reoccurrence\_freq\_units* | 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 376 for more information about how to send messages to OVO.

Table 4‑99 SendAlarm Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| module | Yes | Name of a module that will take care of sending the message. | None | String |
| message | 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 constant:*X*. The message can contain free variables (indicated by %s symbol) to be replaced with other parameters. | None | String |
| param0, param1... paramN | No | Names of case-packet variables whose values replace the free variables in the message.  Specify as many param 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.

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| severity | No | Set the severity of the message that is being sent to OVO. This overrides the default severity set in the OVOMessageModule configuration. | None |  |
| application | 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 |  |
| Object | No | Set the name of the object on whose behalf the message will be sent.  This overrides the default (blank). | None |  |
| msg\_grp | 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 |  |
| Node | No | Set the name of the node on whose behalf the message will be sent.  This overrides the default (null). | None |  |
| service\_id | No | Set the service\_id for the message that is being sent to OVO. This overrides the default (blank). | None |  |

**Example: 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>

### SendMail

com.hp.ov.activator.mwfm.component.builtin.notification.mail.SendMail

This node uses the MailSenderModule in order to send emails in an asynchronous way. In this context, *asynchronous* means that the workflow node will not keep waiting for the mail delivery to the mail server.

**NOTE** The email could be rejected afterwards due to several reasons, such as wrong recipient, exceeding size limitation, and so on.

Itis required to configure the module beforehand by specifying the connection parameters, such as the email server hostname, the email server port, and so on. For more details, see *Section 6.24 MailSenderModule*.

**NOTE** Itis mandatory to specify at least one of the following input parameters: *to*, *to*\_*list*, *bcc*, *bcc*\_*list*, *cc*, or *cc\_list*.

Table 4‑100 SendEmail Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| module\_name | Yes | The name of the associated MailRequestModule | None | String |
| body | Yes | The content of the message | None | String |
| subject | Yes | The subject of the message | None | String |
| to0 to1 … toN | No | The email address of the email’s recipient. | None | String |
| to\_list | No | A list of addresses of the email’s recipient. | None | List |
| cc0 cc1 … ccN | No | The carbon copy email address. | None | String |
| cc\_list | No | A list of carbon copies. | None | List |
| bcc0 bcc1 … bccN | No | The blind carbon copy email address. | None | String |
| bcc\_list | No | A list of blind carbon copies. | None | List |
| from | No | The email address of the sender | None | String |
| mail\_format | No | It defines the format of the content: TEXT or HTML | TEXT | String |
| priority | No | It supports three levels: 5 – Low, 3 – Normal, 1 – High . The value will be included in the mail header “X-Priority” | None | Number |
| attachment0 attachment1 … attachmentN | No | A list of filenames to be attached to the message. The files must be located in the folder: *$ACTIVATOR\_VAR*/mail | None | List |
| message\_id | Optional | A case packet variable that will contain the message identifier returned by the server | None | Object |

**If the operation throws an exception, then the node will return -1 as return value. Else it will return zero and will set the** message\_id **if required.**

**Example: SendEmail - use in the workflow**

This example assumes that the target variable contains a text with the email’s recipient and the module is also well configured.

<Process-Node disablePersistence="true">

<Name>SendMail</Name>

<Action>

<Class-Name>com.hp.ov.activator.mwfm.component.builtin.notification.mail.SendMail</Class-Name>

<Param name="body" value="constant:We&apos;ll release a new version very soon."/>

<Param name="from" value="constant:team.hpsa@hpe.com"/>

<Param name="module\_name" value="constant:mail\_sender\_module"/>

<Param name="subject" value="constant:New release is coming ..."/>

<Param name="to0" value="target"/>

</Action>

</Process-Node>

</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 398 for more information about how to send messages to a waiting program.

Table 4‑101 SendMessage Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| skip | No | If true this node will do nothing | false | Boolean |
| sender | Yes | Name of the Workflow Manager module that will send this message. | None | String |
| message\_var | Yes, if message\_url is not used | Name of a case-packet variable that contains the message to be sent. If both "sor\_id" and "message\_var" are passed to "SendMessage" then the value in "message\_var" is actually sent, whereas the "response\_config" corresponding to the SOR record identified by "sor\_id" is used to find out "where" to send the message. | None | String |
| message\_url | Yes, if message\_var 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 db:message\_id or file: file\_path. A constant file name or message id can also be specified. | None | String |
| use\_sor\_id | No | Set this parameter to “true” to read the message from the Service Order Registry, using “SOR\_ID” as identifier. The state of the entry in the Service Order Registry will be updated before and after sending the message. NOTE: This parameter can only be used if the sender module is using database to save the send message in if not possible to send the message at the moment. This parameters is mutually exclusive with “sor\_id” parameter. | false | Boolean |
| sor\_id | No | Populate this parameter to explicitly identify the SOR record to use by the node/handler (unlike use\_sor\_id which -- if set to true -- uses the built-in SOR\_ID). This parameters is mutually exclusive with “use\_sor\_id”. | None | Integer |
| state\_finished | No | Set this parameter to “true” to set state to FINISHED in the Service Order Registry when the message has been sent successfully. if set to false then the state after sending the message would be INCOMPLETE. The state of the entry in the Service Order Registry will be updated before and after sending the message, but only if use\_sor\_id is set to true. NOTE: This parameter can only be used if the sender module is using database to save the send message in if not possible to send the message at the moment. | false | Boolean |
| throw\_excep | No | Controls whether the node should throw exceptions upon failures, or if the framework should handle them. If set to 'false' the framework will handle the failure by setting the RET\_VALUE case packet variable to -1. The RET\_TEXT variable will hold the failure text. | true | Boolean |

**Example: 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: 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: 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"/>

</Case-Packet>

<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 SNMPSenderModule. The SNMPSenderModule 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 4‑102 SendSNMPTrap Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| Module | Yes | The SNMP Sender module to be used | None | String |
| trap\_oid | Yes | Oid of the trap | None | String |
| oid0,  oid1,  ... oidN | No | Additional oid to include in the trap | None | String |
| trap\_message, trap\_message1,  ...  trap\_messageN | No | The trap message associated with the oid | None | String |

**Example: 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 4‑103 Sleep Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| time | 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 constant:*X*. | 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: 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>

</Process-Node>

### SORGetData

com.hp.ov.activator.mwfm.component.builtin.SORGetData

Reads the data for a specific service order registry record. The record can be found eiter by providing the system name and transaction id or providing the sorId.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| system\_name | If sor\_id is not provided | The system name for this service order registry record. | None | String |
| transaction\_id | If sor\_id is not provided | The transaction id for this service order registry record. | None | String |
| sor\_id | If system\_name and transaction\_id are not provided | The id of the service order registry record. | None | String |
| sor\_data | Yes | The case-packet which will contain the service order registry record after the workflow execution. | None | Object |

### SORSaveMessage

com.hp.ov.activator.mwfm.component.builtin.SORSaveMessage

Updates an entry in Service Order Registry with the given values.

Convert the response object to a string by calling the specified bean and method. In case a class and method is not specified then the object must be a String otherwise the node will fail. The response object and string is saved to Service Order Registry.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| skip | No | If true this node will do nothing | false | Boolean |
| skip\_save | No | Causes the node/handler to work as normal, except that it does not store anything to SOR (unlike the skip parameter that can be used to completely skip the node/handler). | false | Boolean |
| sor\_id | Yes | Identifier of the SOR entry. | None | Integer |
| send\_workflow | Yes | The workflow to be called in case retry of sending the response should be done after the primary response is sent | None | String |
| message\_object | Yes | The message object which represent what is going to be the result | None | Object |
| message\_string | No | When the node is executed this case-packet variable is set to what is the output of the method call or directly what is the value in messageObject in case no class and method is defined | None | String |
| Bean | No | If constant: Call static method on this (fully qualified) class name. If variable: Invoke the named method on the passed object. Pass the DB connection if required. | None | Object |
| Method | If bean is specified | The name of the method to be invoked. | None | String |
| sor\_state | No | The state the SOR record should be updated to. | RUNNING | String |
| message\_type | Yes | The message type of the SOR record. | None | String |
| operation\_name | Yes | The name of the operation related to this SOR entry. | None | String |
| customer\_name | Yes | The name of customer to which the operation associated to this SOR entry is related. | None | String |
| process\_id | No | The process identifier. If not provided, the existing one will remain unmodified. | None | String |
| service\_id | No | The identifier of the service to which this SOR entry is related. If not provided, the existing one will remain unmodified. | None | String |
| create\_record | No | Setting it to true will create a new SOR record. This should be enabled for specific cases where it’s not required to start a WF. | false | Boolean |
| username | No | (If create\_record is true) it sets the username that has requested the action | None | String |
| system\_name | Yes, if create\_record is true | (If create\_record is true) it sets the system name. | None | String |
| order\_id | No | (If create\_record is true) it sets the order identifier. | None | String |
| synchronous | Yes, iif create\_record is true | (If create\_record is true) it sets if the request is synchronous or not. | false | Boolean |
| input\_object | No | (If create\_record is true) it contains the request input object. | None | Object |
| response\_config\_display | No | (If create\_record is true) it sets the display type for the response configuration. | None | String |
| result\_description | No | (If create\_record is true) it contains a description for the operation result. | None | String |
| input\_message | No | (If create\_record is true) it contains the input message. | Empty | String |
| transaction\_id | Yes, if create\_record is true | (If create\_record is true) it contains the required transaction identifier for the request. | None | String |

### 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 81 for more information about how one workflow can wait for another

Table 4‑104 StartJob Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| workflow\_name | 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 constant:*X*. | None | String |
| variable0, variable1... variableN | No | Names of case-packet variables to be passed to the child workflow. | None | Object |
| destination0, destination1... destinationN | 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.  The system case packet Boolean variable DISABLE\_PERSISTENCE, can be used to disable the persistence of a persistent workflow. | None | Object |
| PRIORITY | No | Name of a case-packet variable that contains the priority of the workflow to be started. To specify a hard-coded value, use the syntax constant:*X*. | The parent’s current priority | Integer |
| output\_job\_id\_variable | No | A case-packet variable to catch the JOB\_ID of the newly started job | None | Integer |

**Example: 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 235

Table 4‑105 StartJobAndWait Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| workflow\_name | Yes | Name of the workflow to start. | None | String |
| variable0, variable1... variableN | No | Names of case-packet variables to be passed to the child workflow. | None | Any |
| destination0, destination1... destinationN | No | The name of the case-packet variable to initialize in the new workflow. By default the variable of the same name is initialized.  The system case packet Boolean variable DISABLE\_PERSISTENCE, can be used to disable the persistence of a persistent workflow. | None | Any |
| outputvar0, outputvar1... outputvarN | No | Case-packet variables that the child should pass back. | None | Object |
| PRIORITY | No | Name of a case-packet variable that contains the priority of the workflow to be started. To specify a hard-coded value, use the syntax constant:*X*. | The parent’s current priority | Integer |
| queue | 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: - 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>

### StartQueueJob

com.hp.ov.activator.mwfm.component.builtin.StartQueueJob

The node starts a new job in the specified queue 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 81 for more information about how one workflow can wait for another

Table 4‑106 StartQueueJob Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| workflow\_name | 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 constant:*X*. | None | String |
| solution\_name | No | Name of the solution used to identify the queue where to start the workflow. | None | String |
| queue\_name | Yes | Name of the queue where to start the workflow. | None | String |
| variable0, variable1... variableN | No | Names of case-packet variables to be passed to the child workflow. | None | Object |
| destination0, destination1... destinationN | 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.  The system case packet Boolean variable DISABLE\_PERSISTENCE, can be used to disable the persistence of a persistent workflow. | None | Object |
| PRIORITY | No | Name of a case-packet variable that contains the priority of the workflow to be started. To specify a hard-coded value, use the syntax constant:*X*. | The parent’s current priority | Integer |
| output\_job\_id\_variable | No | A case-packet variable to catch the JOB\_ID of the newly started job | None | Integer |

**Example: StartQueueJob - use in the workflow**

This example starts a child workflow in the S1#Q1 queue 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.StartQueueJob

</Class-Name>

<Param name="workflow\_name" value="workflow"/>

<Param name="solution\_name" value="S1"/>

<Param name="queue\_name" value="Q1"/>

<Param name="variable0" value="JOB\_ID" />

<Param name="destination0" value="controller\_job\_id" />

<Param name="variable1" value="message\_file" />

</Action>

</Process-Node>

### 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 varaible 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 4‑107 Switch Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| key | Yes | The key that is evaluated and which decides the workflow path to taken | None | String or Integer |
| case0 case1... caseN | Yes | Case values specified for various branches. | None | Depend on the bean |

**Example: 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 nod e 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 81 for information about how one workflow can wait for input from another workflow

Table 4‑108 Sync Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| job\_id | Yes | Name of a case-packet variable that contains the job ID of the workflow waiting to synchronize. | None | Integer |
| queue | Yes | Name of the queue on which the workflow is waiting, specified as a constant string. | None | Queue |
| OK | No | Name of the Boolean  case-packet variable to catch the indication of whether the Sync was successful or not. | None | Boolean |
| variable0, variable1... variableN | 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. destinationN parameters can be specified selectively for some or all of the indicated variables. | None | Any |
| destination0, destination1... destinationN | 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: 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 4‑109 ThrowError Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| Message | 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 4‑110 ThrowException Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| message | 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 4‑111 ThrowRuntimeException Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| message | 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 4‑112 TransformXML Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| xml\_url | 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 |
| xml\_var | no | The name of a case-packet variable from which to get the XML input document. | None | String |
| xsl\_url | either xsl\_url or xsl\_var | 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 |
| xsl\_var | either xsl\_url or xsl\_var | The name of a case-packet variable from which to get the XSL specification. | None | String |
| output\_url | 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 |
| output\_var | 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\_f iles 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 operation system | String |

**Example: 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: 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/Tran](http://www.w3.org/1999/XSL/Transform)sform" 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: 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 Workflows Data Types in Workflow Node Parameters” on page 39 with the modification that the special characters must be substituted as shown in Table 4‑113.

Table 4‑113 Complex Data Type Usage in XSLT

|  |  |
| --- | --- |
| Symbol | Replacement in xslt |
| { | \_LC\_ |
| } | \_RC\_ |
| [ | \_LB\_ |
| ] | \_RB\_ |
| “ | \_Q\_ |
| # | \_H\_ |

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 arrayCP\_\_LB\_\_0\_\_RB\_\_, where the character “{“ is replaced by \_\_LB\_\_ and the character “]” is replaced by \_\_RB\_\_ .
* The element hashMapcp {"key1"} is written as hashMapcp\_\_LC\_\_\_\_Q\_\_key1\_\_Q\_\_\_\_RC\_\_ , 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 $hashMapcp\_\_LC\_\_\_\_Q\_\_key1\_\_Q\_\_\_\_RC\_\_ , $arrayCP\_\_LB\_\_0\_\_RB\_\_ and $beanCP\_\_D\_\_attribute1.

<?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="arrayCP\_\_LB\_\_0\_\_RB\_\_"/>

<xsl:param name="beanCP\_\_D\_\_attribute2\_\_LB\_\_0\_\_RB\_\_"/>

<xsl:param name="hashMapcp\_\_LC\_\_\_\_Q\_\_key1\_\_Q\_\_\_\_RC\_\_"/>

<xsl:param name="hashMapcp\_\_LC\_\_beanCP\_\_D\_\_attribute2\_\_LB\_\_0\_\_RB\_\_\_\_RC\_\_"/>

<xsl:param

name="hashMapComplexcp\_\_LC\_\_beanCP\_\_D\_\_attribute2\_\_LB\_\_0\_\_RB\_\_\_\_RC\_\_\_\_LC\_\_a

rrayCP\_\_LB\_\_1\_\_RB\_\_\_\_RC\_\_\_\_D\_\_attribute2\_\_H\_\_"/>

<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;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 4‑114 UpdateBean Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| bean\_object | Yes | Name of the variable holding the inventory bean object to be updated. | None | Object |
| key\_field0, key\_field1... key\_fieldN | 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 bean\_object parameter there is no need for specifying the primary key | None | String |
| key\_value0, key\_value1, key\_valueN | Yes | Used in conjunction with the key\_field attributes to specify the values of the individual attributes in the JavaBean | None | Any |
| bean\_variable | Yes | Name of the variable where the created JavaBean instance is returned. | None | Object |

**Example: 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 4‑115 UpdateCustomAttributesNNMNode Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| module\_name | Yes | The name of the Micro Focus NNMi\* module used to connect to a specific Micro Focus NNMi\* server | None | String |
| bean\_type | 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 |
| bean\_id | Yes | The the identifier of the bean whose custom attributes will be updated. | None | String |
| action | Yes | The parameter determines which operation is to be performed: The value can be either "ADD" or "REMOVE". | None | String |
| custom\_attribute\_name0, custom\_attribute\_name1, ... custom\_attribute\_nameN | No | Custom attribute name | None | String |
| custom\_attribute\_value0, custom\_attribute\_value1,  ... custom\_attribute\_valueN | No | Custom attriubte value | None | String |
| custom\_attribute\_map | Yes, If no custom\_att ribute\_name 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 4‑116 UpdateInProgress Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| service\_id | Yes | Unique identifier that the technical parameters are bound to. | None | String |
| db | No | Name of the database pluggable module to be used. | "db" | String |
| status | Yes | New status value to set the IN\_PROGRESS 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: 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 primar y 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 4‑117 UpdateInventory Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| db | No | Name of the database module to be used. | “db” | String |
| bean | Yes | Name of the JavaBean class that is used for storing the data. | None | String |
| bean\_object | No | The name of the variable containing the inventory bean object to by stored in the inventory. | None | Object |
| key\_field0, key\_field1... key\_fieldN | No | Name of a key in the JavaBean that is updated or created. The parame- ter 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 ofkeys, even if it is not updated. Note that the key\_fields may be case packet variables. | None | String |
| key\_value0, key\_value1, key\_valueN | No | Used in conjunction with the *key\_field* attributes to specify the new value of the individual attributes in the JavaBean | None | Depends on the bean |
| bean\_variable | No | Name of the variable where the created/updated JavaBean instance is returned. | None | Object |
| strict\_create | 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 *strict\_update* parameter. | false | Boolean |
| strict\_update | 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 *strict\_create* parameter. | false | Boolean |
| store\_audit | 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: UpdateInventory - use in the workflow**

This example uses the UpdateInventory noode to modify the state of a port. The primary key is held in the variable port\_idand 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>

### UpdateManagementModeNNMNode

com.hp.ov.activator.mwfm.component.builtin.nnmrequest.UpdateManagementModeNNMNode

The node changes the Management Mode of an NNMi Node to the new desired value.

Table 4‑118 UpdateServiceInstance Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| module\_name | Yes | Name of the NNMi Module to use. | None | String |
| node\_id | Yes | NNMi identifier of the Node to update. | None | String |
| management\_mode | Yes | New NNMi Management Mode that the node should be set to: “INHERITED”, “MANAGED”, “NOTMANAGED”, “OUTOFSERVICE” | None | String |

**Example: UpdateManagementModeNNMNode- use in the workflow**

This example sets a node management mode to “MANAGED”.

<Process-Node>

<Name>UpdateManagementModeNNMNode</Name>

<Description></Description>

<State></State>

<Action>

<Class-Name>com.hp.ov.activator.mwfm.component.builtin.nnmrequest.UpdateManagementModeNNMNode</Class-Name>

<Param name="management\_mode" value="constant:MANAGED"/>

<Param name="module\_name" value="nnmiModule"/>

<Param name="node\_id" value="nodeId"/>

</Action>

</Process-Node>

### 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 4‑119 UpdateServiceInstance Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| service\_id | Yes | Unique identifier to which the technical parameters are bound. | None | String |
| db | No | Name of the database module to use. | “db” | String |
| variable0, variable1, … | Yes | Names of the different technical parameters to update. You must specify at least *variable0*. | None | String |

**Example: 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>

### 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 the Assign node or the MultiAssign node.

**See Also**

* “Assign” on page 86
* “MultiAssign” on page 179

Table 4‑120 VariableMapper Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| template\_file | 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.  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 |
| use\_solution\_dir | No | When set to "true", the nodes will read from $SOLUTION\_ETC/template\_files instead of $ACTIVATOR\_ETC/template\_file s. | false | boolean |

**Example: 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 4‑121 WasPreviousNodeOK Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| priv\_ret\_value | No | This argument makes it possible to save the value of the system case-packet variable RET\_VALUE for the previous node. This value may be necessary to handle errors later on. | None | Integer |
| priv\_ret\_text | No | This argument has the same functionality as the above apart from saving the RET\_TEXT variable. | None | String |

**Example: 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 4‑122 WriteCasePacket Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| File | 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 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 | String |
| Sender | Yes, if file 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 identifer must be written to the identifer 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 4‑123 WriteDataToDatabase Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| message\_url | No | Name of the case packet variable holding the message id. The syntax is db:<message\_id> | None | String |
| identifier | No | Name of the case packet variable holding the identifier. The information will be written in the identifier column | None | String |
| output\_value | Yes | Case packet variable holding the returned message id. The syntax is db:<message\_id>. | None | String |
| message\_data | Yes | Name of the case packet variable containing the message to be written. | None | Object or String |
| data\_position | No | The value of this parameter is the ordinal position from where the data is written. | None | String |

**Example: 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: 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>

### WSComposeRequest

com.hp.ov.activator.mwfm.component.builtin.WSComposeRequest

The node receives a set of parameters and generates a textual XML. It acts like a helper node and is mainly combined with the GenericWS plugin..

The connector parameters must comply the JbossWS requirements. In the same way, the secure connections are configured throught these parameters according with the official documentation: <https://docs.jboss.org/author/display/JBWS/WS-Security#WS-Security-Client>

Table 4‑124 WSComposeRequest Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| wsdl | Yes | The location for the associated WSDL for the web service. If it’s in the local filesystem, it must start with the prefix: ‘file:/’ | None | String |
| operation | Yes | The method or operation to be invoked. | None | String |
| descriptor | Yes | It defines where is located the associated descriptor for the WS. It must start with the prefix: 'db:' or 'file:'. If it is saved in the database (because it was deployed with the Deployment Manager), then the parameter solution\_name is mandatory. If it located in the filesystem, then it can be locally resolved in two ways:  - if it contains only the filename then it will be in the folder: $ACTIVATOR\_ETC/config/ws  - or it can use an absolute path. | None |  |
| solution\_name | Yes (if descriptor is in the db) | It indicates to what solution belongs. | None | String |
| output | Yes | Case packet variable that will hold the output XML | None | String |
| username | No | Username if the authentication is required | None | String |
| password | No | Clear password if the authentication is required | None | String |
| encrypted\_password | No | Encrypted password (if this parameter is defined then the ‘password’ must not be used). | None | String |
| reuse\_connection | No | Defines if the internal instance can be reused (Please read the GenericWS plugin appendix for more information). | ‘false’ | String |
| response\_mode | No | It defines how the response data is formatted:   * CLASSIC: a map will contain only those required output parameters. * FULL: a map will contain all the output parameters that have been defined in the descriptor for that operation. * XML: a textual XML with the RAW response. | CLASSIC | String |
| input\_param0, input\_param1 … | No | It contains the key to identify a property of the request. Usually this key will be an alias or path defined in the WS descriptor. | None | String |
| input\_variable0, input\_variable1… | No | It contains the value for the associated property. | None | String |
| output\_param0, output\_param1… | No | It contains the key to be mapped out for the response. | None | String |
| output\_variable0, output\_variable1… | No | It defines in which variable must be saved the value of one property of the response. | None | String |
| connector\_param0, connector\_param1… | No | It contains the key for the parameter to be used by the WS endpoint implementation class. | None | String |
| connector\_variable0, connector\_variable1… | No | It contains the value for the parameter to be used by the WS endpoint implementation class. | None | String |
| input\_map | No | It contains a map of key-value pairs that will be used as input parameters. It’s similar to the input\_paramX and input\_variableX and both types can be used at the same time. | None | Object |
| output\_map | No | It contains a map of key-value pairs that will be used as output parameters. It’s similar to the output\_paramX and output\_variableX and both types can be used at the same time. | None | Object |
| store\_always | No | Indicate if the exchanged messages have to be always saved to the DATABASE\_MESSAGE table. | None | Boolean |
| store\_failure | No | Indicate if the exchanged messages have to be saved to the DATABASE\_MESSAGE table when an error happens during the communication. | None | Boolean |
| identifier | No | If store\_always or store\_failure are set to true, then the value of the identifier will be set to the identifier column in the DATABASE\_MESSAGE and used as key for the messageId in the uploaded\_data map. | GenericWS | String |
| handler\_classname0, handler\_classname1… | No | It contains the canonical classnames for custom handlers that can intercept the exchanged SOAP messages and faults. | None | String |
| header\_param0,  header\_param1 … | No | It contains a key to identify an entry header of the transport layer. | None | String |
| header\_variable0,  header\_variable1 … | No | It contains a key to identify the associated value for a header of the transport layer. | None | String |
| transport\_headers\_map | No | It contains a map of key-value pairs that will be used to set the headers to the transport layer. | None | Object |

**NOTE: Setting headers to the transport layer could cause unexpected effects if the reuse\_connection is enabled. The headers are associated to the connection object so if a request sets a header in the connection object, the same header will be present for the next requests that reuse the same connection.**

Example WSComposeRequest

<Process-Node>

<Name>WSComposeRequest</Name>

<Action>

<Class-Name>

com.hp.ov.activator.mwfm.component.builtin.WSComposeRequest

</Class-Name>

<Param name="descriptor" value="db:descriptor.xml" />

<Param name="solution\_name" value="tv" />

<Param name="operation" value="queryLocation" />

<Param name="wsdl" value="file:/C:/HP/OpenView/ServiceActivator/etc/config/ws/tv.wsdl" />

<Param name="username" value="admin" />

<Param name="password" value="secretPasswd" />

<Param name="input\_param0 " value="QueryLocation.locationQuery.name" />

<Param name="input\_variable0" value="LocationNameVar" />

<Param name="input\_param1 " value="Holder.version" />

<Param name="input\_variable1" value="tvVersionVar" />

<Param name="output\_param0 " value="location.#0.id" />

<Param name="output\_variable0" value="locationId" />

<Param name="reuse\_connection" value="true" />

</Action>

</Process-Node>

#### Custom handlers

This feature enhances the flexibility for the messages and fault handling. Sometimes customers need to inspect which is the request or the fault returned by the external service. Using a custom handler is possible to intercept the information just before to be sent/received what enables users to modify the request, log the request/response … The nodes WSComposeRequest and WSRequest let configure the handlers via the parameters: handler\_classname0, handler\_classname1…

The HPE Service Activator defines internally a custom handler in order to log the messages to the database when the flags store\_always or store\_failure are set to true. It’s important to note that a wrong implementation could cause side effects even a system malfunction. In this sense it’s recommended to follow the interface definition and implement lightweight operations that cannot degrade the performance.

To create your custom handler:

1. Create your custom Java class that implements next interface: javax.xml.ws.handler.soap.SOAPHandler
2. Compile it and package it into a JAR File.
3. Copy the JAR file to javax.xml.ws.handler.soap.SOAPHandler
4. Restart the HPE Service Activator and set the new complete classname in the node parameters: handler\_classname0, handler\_classname1…
5. Deploy the modified workflows and reload them (if were previously deployed).

Note the JAR file must exist in all the HPE Service Activator cluster nodes.

Please see the Java API for further details about the javax.xml.ws.handler.soap.SOAPHandler

### WSRequest

com.hp.ov.activator.mwfm.component.builtin.WSRequest

The node processes WS requests and requires one associated module.

Although the node supports the configuration of the “url”, “username” and “password”; it is not recommended because the connection is instantiated by request (what’s slower than define the same configuration in the module).

Table 4‑125 WSRequest Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| module | Yes | The associated WSModule that is going to manage the connection with the external system. | None | String |
| operation | Yes | The method to be invoked. | None | String |
| url | No | By default it’s configured in the module what’s faster than the node configuration. | None | String |
| username | No | By default it’s configured in the module. It can be used only when the url is defined in the node. | None | String |
| password | No | By default it’s configured in the module. It can be used only when the url is defined in the node. | None | String |
| encrypted\_password | No | By default it’s configured in the module. It can be used only when the url is defined in the node. | None | String |
| throw\_excep | No |  | true | String |
| response\_mode | No | Change the format of the response:   * CLASSIC : The response is mapped out to workflow variables, depending of the mapping: output\_param0, output\_param1... * FULL: all the values of the response will be extracted to the response map. It makes sense to use when the parameter “return\_map” is true because that map will contain the complete response. * XML: The RET\_TEXT will contain the text representation of the response. | CLASSIC | String |
| input\_param0, input\_param1 … | No | It contains the key to identify a property of the request. Usually this key will be an alias or path defined in the WS descriptor. | None | String |
| input\_variable0, input\_variable1… | No | It contains the value for the associated property. | None | String |
| output\_param0, output\_param1… | No | It contains the key to be mapped out for the response. | None | String |
| output\_variable0, output\_variable1… | No | It defines in which variable must be saved the value of one property of the response. | None | String |
| return\_map | No | If it is true, then the response object will be a map that will be located in the case packet variable defined in the parameter: “response\_map”. Besides the mapping defined for the “output\_param” won’t be used. | false | String |
| response\_map | No | It indicates what case packet variable will hold the map with the response. | None | Object |
| input\_map | No | It could hold a map with a set of input parameters for the requests. It is a complement for the variables: “input\_param” | None | Object |
| handler\_classname0, handler\_classname1… | No | It contains the canonical classnames for custom handlers that can intercept the exchanged SOAP messages and faults. See more details in the section 4.1.124.1 | None | String |
| header\_param0,  header\_param1 … | No | It contains a key to identify an entry header of the transport layer. | None | String |
| header\_variable0,  header\_variable1 … | No | It contains a key to identify the associated value for a header of the transport layer. | None | String |
| transport\_headers\_map | No | It contains a map of key-value pairs that will be used to set the headers to the transport layer. | None | Object |
| transport\_headers\_response | No | It defines a case packet variable where to save the headers of the transport layer that are extracted from the response. | None | Object |

**NOTE: Setting headers to the transport layer could cause unexpected effects if the reuse\_connection is enabled in the module configuration. The headers are associated to the connection object so if a request sets a header in the connection object, the same header will be present for the next requests that reuse the same connection.**

Example WSRequest

<Process-Node>

<Name>WriteDataToDatabase</Name>

<Action>

<Class-Name>com.hp.ov.activator.mwfm.component.builtin.WSRequest</Class-Name>

<Param name="module" value="trueview\_ws\_module"/>

<Param name="operation" value="queryLocation"/>

<Param name="response\_mode" value="CLASSIC"/>

<Param name="input\_param0" value="QueryLocation.locationQuery.name"/>

<Param name="input\_variable0" value="locationNameVar"/>

<Param name="input\_param1" value="Holder.version"/>

<Param name="input\_variable1" value="ws\_request\_version"/></Action>

<Param name="output\_param0" value="location.#0.id"/>

<Param name="output\_variable0" value="locationId"/>

</Process-Node>

### WriteFile

com.hp.ov.activator.mwfm.component.builtin.WriteFile

The node writes the contents of a case packet variable into a text file.

Table 4‑126 WriteFile Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Required | Description | Default | Type |
| File | 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 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 read relative to *$ACTIVATOR\_VAR*. | None | String |
| variable | Yes | Name of the case-packet variable with the contents that must be written into the file. | None | String |
| Charset | No | The charset to be used when converting retrieved data to a String. | None | String |

### 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 4‑127 XMLMapper Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| xml\_url | Yes (if not xml\_var is specified) | The file to be mapped (an absolute path, or a filename relative to $ACTIVATOR\_VAR), or the message id that refers to a message stored in the database, or a data string. The syntax is file:<file path>, data:<string>, or db:<message\_id>. | None | String |
| template\_file | No | Name of a file that holds a list of mappings. The default path to find files is:  *$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 |
| validate | 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 |
| ignore\_missing\_tags | 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 |
| xml\_var | Yes (if not xml\_url is specified) | The name of the case-packet variable containing the XML message. | None | String |
| preserve\_variable\_index | No | Indicating if wheater an optional element having been missed out should be represented by a null value | False | Boolean |
| use\_solution\_dir | No | When set to "true", the nodes will read from  $SOLUTION\_ETC/templ ate\_files instead of  $ACTIVATOR\_ETC/templ ate\_files. | false | Boolean |

**Example: 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: 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: XMLNamespaces**

<?xml version=”1.0” encoding=”UTF-8”?>

<Document

xmlns:ns1=["http://www.hp.com/ns1"](http://www.hp.com/ns1) xmlns:ns2=["http://www.hp.com/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 4‑128 XMLParser Parameters

| Name | Required | Description | Default | Type |
| --- | --- | --- | --- | --- |
| *xml\_var* | Yes (if message\_url is not given) | The case packet variable which contains the xml | None | String |
| message\_url | 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 |
| validation | No | Indicates whether the xml needs to be validated against a DTD/Schema | True | Boolean |
| definition\_language | No | Indicates the type of language used to validate the xml. Possible values are DTD and W3CSchema | W3CSchema | String |
| definition\_url | No | The url to locate the schema or the DTD(an absolute path, or a filename relative to $ACTIVATOR\_ETC/confi g), The syntax is file:<file path>. | None | String |
| template\_file | No | Name of a file that holds a list of mappings. The default path to find files is: *$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 |
| use\_solution\_dir | No | When set to "true", the nodes will read from $SOLUTION\_ETC/templ ate\_files instead of $ACTIVATOR\_ETC/templ ate\_files. | false | Boolean |
| 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 namespac e\_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. |  |  |
| ignore\_missing\_tags | 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 HPE 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 HPE 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 91

Table 4‑129 ComposeMessageHandler Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| template\_file | Yes if template\_var 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 use\_solution\_dir parameter. | None |
| template\_var0, template\_var1,  ... template\_varN | Yes if template\_file is not used | Name of a case-packets available that contains the template strings. | None |
| use\_solution\_dir | No | When set to true,the handler will read from $SOLUTION\_ETC/template\_files instead of $ACTIVATOR\_ETC/template\_file | false |
| output\_file | Yes if output\_var 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 |
| output\_var | Yes if output\_file 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 179

Table 4‑130 MultiAssignHandler Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| variable0, variable1,  ... variable*N* | Yes | Case-packet variables to be set. | None |
| value0, value1,  ... value*N* | Yes | 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 |
| default0, default1,  ... default*N* | No | Default value to be used if the corresponding value is null, empty, or 0. | 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 109

Table 4‑131 DoNothingHandler Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| Message | No | The message to be logged. | None |

### MethodHandler

com.hp.ov.activator.mwfm.component.builtin.MethodHandler

Invokes a method on a Java class (static) or object (instance) such as an inventory bean. If the method requires an SQL connection, a connection argument is automatically added and passed to the method. The node attempts to do conversion of argument and return types when possible.

**See Also**

* “InvokeMethod” on page 146
* “MethodRule” on page 174

Table 4‑132 MethodHandler Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| db | No | Name of the module to access the database if the method needs it. | “db” |
| bean | Yes | If constant: Call static method on this (fully qualified) class name. If variable: Invoke the named method on the passed object. | None |
| method | No | The name of the method to be invoked. | findByPrimaryKey |
| arg0, arg1, ... arg*N* | No | The arguments to be passed to the method. Do not explicitly pass any SQL Connection arguments, as they are passed automatically | None |
| variable | No | The name of a case packet variable to catch the method return value if any. This parameter is optional - do not define it when calling void-type methods. | 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 th 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.

See Also

* “PutMessage” on page 191

**NOTE** If the message is more than 4000 bytes the message will be truncated to 4000 bytes.

Table 4‑133 PutMessageHandler Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| queue | Yes | Queue where the message is left. This parameter can either be a constant or a case-packet variable. Spaces are not allowed. | None |
| message | 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 |
| param0,  param1,  ... paramN | 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 |
| service\_id | No | The Service Identifier value used for associating the message with a service or a solution. | None |
| order\_id | No | The order Identifier value used for associating the message with a service or a solution. | None |
| type | No | The type value of the workflow | None |
| state | No | The state value of the workflow | None |
| skip | No | If true then the message will not be written | false |

### 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 95
* “ReleaseResource” on page 210
* “ReserveResource” on page 216

Table 4‑134 ReleaseResourceHandler Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| db | No | Database module to use in order to perform the update. | “db” |
| variable0, variable1... variableN | 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 *RESERVATIONS* variable are released. | None |

### QueueNotificationModuleHandler

Calls back Queue Notification Module when a message has been successfully sent, so Queue Notification Message acknowledges the result of that operation.

This handler does not receive parameters, but requires that the next case packet variables exist.

Table 4‑135 QueueNotificationModuleHandler case packet variables required.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| JOB\_ID | Yes | The job identifier. | None |
| error\_code | Yes | Value 0 indicates no error. Value 1 indicates connectivity issues. Any other value indicates an error sending the message. | None |
| error\_text | Yes | A description indicating the error, if any. | None |
| RET\_VALUE | Yes | If error\_code has been set to 0 (success) then this variable can be used to notify a secondary type of error. The values returned here have the same meaning as described for error\_code. As a consequence, if error\_code is set to 0 but RET\_VALUE is set to 1, connectivity issues will be assumed. | None |
| RET\_TEXT | Yes | A description indicating the error, if any, in case the error is being notified through RET\_VALUE instead of error\_code. | None |
| queue | Yes | The queue type (not the queue name) of the queue(s) to be notified in Queue Notification Module. | None |

### SendMessageHandler

com.hp.ov.activator.mwfm.component.builtin.SendMessageHandler

This handler implements 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 227

Table 4‑136 SendMessageHandler Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| skip | No | If true this node will do nothing | false |
| sender | Yes | Name of the Workflow Manager module that will send this message. | None |
| message\_var | Yes if message\_url is not used | Name of a case-packet variable that contains the message to be sent. If both "sor\_id" and "message\_var" are passed to "SendMessage" then the value in "message\_var" is actually sent, whereas the "response\_config" corresponding to the SOR record identified by "sor\_id" is used to find out "where" to send the message. | None |
| message\_url | Yes if message\_var 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 db:message\_id or file:filepath. A constant file name or message id can also be specified. | None |
| use\_sor\_id | No | Set this parameter to “true” to read the message from the Service Order Registry, using “SOR\_ID” as identifier. The state of the entry in the Service Order Registry will be updated before and after sending the message. NOTE: This parameter can only be used if the sender module is using database to save the send message in if not possible to send the message at the moment. This parameters is mutually exclusive with “sor\_id” parameter. | false |
| sor\_id | No | Populate this parameter to explicitly identify the SOR record to use by the node/handler (unlike use\_sor\_id which -- if set to true -- uses the built-in SOR\_ID). This parameters is mutually exclusive with “use\_sor\_id”. | None |
| state\_finished | No | Set this parameter to “true” to set state to FINISHED in the Service Order Registry when the message has been sent successfully. if set to false then the state after sending the message would be INCOMPLETE. The state of the entry in the Service Order Registry will be updated before and after sending the message, but only if use\_sor\_id is set to true. NOTE: This parameter can only be used if the sender module is using database to save the send message in if not possible to send the message at the moment. | false |
| throw\_excep | No | Controls whether the node should throw exceptions upon failures, or if the framework should handle them. If set to 'false' the framework will handle the failure by setting the RET\_VALUE case packet variable to -1. The RET\_TEXT variable will hold the failure text. | true |

### SORSaveMessageHandler

com.hp.ov.activator.mwfm.component.builtin.SORSaveMessageHandler

Updates an entry in Service Order Registry with the given values.

Converts the response object to a string by calling the specified bean and method. In case a class and method is not specified then the object must be a String, otherwise the node will fail. The response object and string is saved in the Service Order Registry.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| skip | No | If true this node will do nothing | false |
| skip\_save | No | Causes the node/handler to work as normal, except that it does not store anything to SOR (unlike the skip parameter that can be used to completely skip the node/handler). | false |
| sor\_id | Yes | Identifier of the SOR entry. | None |
| send\_workflow | Yes | The workflow to be called in case retry of sending the response should be done after the primary response is sent | None |
| message\_object | Yes | The message object which represent what is going to be the result | None |
| message\_string | No | When the node is executed this case-packet variable is set to what is the output of the method call or directly what is the value in messageObject in case no class and method is defined | None |
| bean | No | If constant: Call static method on this (fully qualified) class name. If variable: Invoke the named method on the passed object. Pass the DB connection if required. | None |
| method | If bean is specified | The name of the method to be invoked. | None |
| sor\_state | No | The state the SOR record should be updated to. Any previous value will be replaced, even if the incominf value is null. | RUNNING |
| message\_type | Yes | The message type of the SOR record. Any previous value will be replaced, even if the incominf value is null. | None |
| operation\_name | Yes | The name of the operation related to this SOR entry. Any previous value will be replaced, even if the incominf value is null. | None |
| customer\_name | Yes | The name of customer to which the operation associated to this SOR entry is related. Any previous value will be replaced, even if the incominf value is null. | None |
| process\_id | No | The process identifier. If not provided, the existing one will remain unmodified. | None |
| service\_id | No | The identifier of the service to which this SOR entry is related. If not provided, the existing one will remain unmodified. | None |
| create\_record | No | Setting it to true will create a new SOR record. This should be enabled for specific cases where it’s not required to start a WF. | false |
| username | No | (If create\_record is true) it sets the username that has requested the action | None |
| system\_name | Yes, if create\_record is true | (If create\_record is true) it sets the system name. | None |
| order\_id | No | (If create\_record is true) it sets the order identifier. | None |
| synchronous | Yes, iif create\_record is true | (If create\_record is true) it sets if the request is synchronous or not. | false |
| input\_object | No | (If create\_record is true) it contains the request input object. | None |
| response\_config\_display | No | (If create\_record is true) it sets the display type for the response configuration. | None |
| result\_description | No | (If create\_record is true) it contains a description for the operation result. | None |
| input\_message | No | (If create\_record is true) it contains the input message. | Empty |
| transaction\_id | Yes, if create\_record is true | (If create\_record is true) it contains the required transaction identifier for the request. | 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 oage 81
* “Sync” on page 242

Table 4‑137 SyncHandler Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| job\_id | Yes | Name of a case-packet variable that contains the job\_id of the workflow waiting to synchronize. | None |
| queue | 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 |
| variable0, variable1... variableN | Yes | Names of the case-packet variables to be passed to the waiting workflow. | None |
| destination0, destination1... destinationN | 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 |
| exception\_destination | No | The value of the exception\_destination parameter indicates the name of a case-packet variable in the target workflow that should receive the exception message. | None |
| sync | 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 259

Table 4‑138 VariableMapperHandler Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Description | Default |
| template\_file | No | Name of a file that holds a list of mappings. The directory to find files is $SOLUTION\_ETC/template\_files or $ACTIVATOR\_ETC/template\_files, depending on the value of the use\_solution\_dir parameter.  You can specify an absolute path name.  Mappings in the file are each on a separate line and have the following syntax var=template | None |
| name of case-packet variable | No | Name of a case-packets available that contains the template strings. | None |
| user\_solution\_dir | No | When set to “true”,the handler will read from $SOLUTION\_ETC/template\_files instead of $ACTIVATOR\_ETC/template\_file | false |

# 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. Workflow Manager modules can also be configured in Workflow Manager configuration snippets as it is described in the next sections.

## 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*

1. 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 303.

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 |
| --- | --- | --- | --- | --- |
| Port | 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 |
| Max-Threads | 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 |
| Min-Threads | 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 Max-Threads setting. Dynamic threads expire after 10 seconds of inactivity, or the time specified in Idle\_Thread\_Keep\_Alive. | Yes. Only increase is allowed, decrease will be ignored until next re-start | Max-Threads |
| Idle\_Thread\_Keep\_Alive | No | The amount of time (in seconds) that an idle dynamic thread exists before it expires and is destroyed (see the Min-Threads discussion) . This only refers to dynamic threads. | Yes | 10 |
| Max-Work-List-Length | No | The maximum number of concurrently running jobs. | Yes | 512 |
| Max-Nodes-Per-Thread | 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 | 5 |
| Test-Mode | No | When set to true the workflow nodes which are marked with the test flag will be executed else they would not. | Yes | false |
| Persistent-Timeouts | 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 | “true” |
| Initial-Workflow-Load | No | If true, on start-up the Workflow Manager attempts to read and validate all of the workflows in the *$ACTIVATOR\_ETC*/ workflows directory. Any errors are logged to the mwfm.stdout log file. | No | “true” |
| Statistics | No | Specifies the maximum workflow runtime history for the statistics manager within a HPE Service Activator workflow. If this parameter is absent, no statistics are performed. Using the statistic manager might slow down the system. | No | None |
| Admin-Role | 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 |
| Queue-Timeout-Seconds | 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 |
| Error-Messages | 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” |
| Permanent-Queue | 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 |
| Java-Class-Path | 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 |
| Dyn-Class-Path | No | Path the dynamic classes created by the java rule and java process nodes | No | None |
| Resource-Manager-RMI-Host | No | The host name where the resource manger is running. Must be set to local host | No | None |
| Resource-Manager-RMI-Port | No | The RMI port for the Resource Manager. | No | None |
| Resource-Manager-RMI-Name | No | The name with which the resource manager RMI interface is bound. | No | None |
| Management-Realm-Username | Yes | The user name for the HTTP connection to the JBoss CLI | No | None |
| Management-Realm-Password | Yes | The password for the HTTP connection to the JBoss CLI | No | None |

**Example: 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: Creating a Permanent Message Queue Viewable by “admin” Users Only**

<Permanent-Queue name="admin" type="message">

<Role>admin</Role>

</Permanent-Queue>

**Example: 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 can also configure modules using Workflow Manger configuration snippets. These snippets are usually related to solutions. Each configuration snippet must be an XML file deployed below $ACTIVATOR\_ETC/config/mwfm and each one must fulfill $ACTIVATOR\_ETC/config/mwfm/solutionmwfm.dtd. If more than one snippet is found then they are loaded in lexicographical order.

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.

Modules configured in snippets can optionally define a dependency with another module. In such case, it is granted that the module will be resumed once the one on which it depends has been resumed.

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 HPE 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”](#_bookmark244) is the built-in Java class to use for this functionality. The XMLLogModule logs messages using an XML format into regular files. The “SolutionXMLogModule” 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 Workflow Manager *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 which, in turn, by default uses the WeightedEngineQueue (supports priorities and is starvation free).

#### 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 cluser node in case this node shut down unexpectedly.

If HPE 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. The “DBTransactionModule” (see page 318) must be used to make use this this functionality.

#### Activation

An activation module provides the interface between the Workflow Manager and the activation engine (Resource Manager). Virtually all installations of HPE Service Activator have one module configured with the name “activator”. The [“ActivationModule”](#_bookmark190) 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 Authenticates by using information saved in the database through the User Management Interface (see “DatabaseAdvancedAuthModule” on page 315).

LDAPAuthModule Authenticates usernames against an LDAP directory server (see “LDAPAuthModule” on page 358).

LinuxAdvancedAuthModule Authenticates Linux usernames (see “LinuxAdvancedAuthModule” on page 362).

WindowsAdvancedAuthModule Authenticates Windows usernames (see “WindowsAdvancedAuthModule” on page 406).

It is also possible to provide your own authentication module. See “Writing New Authenticator Module” on page 433.

#### 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 HPE 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 confiugred 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”](#_bookmark238) is the only built-in Java class to use for this functionality. Every installation of HPE 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 HPE 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 HPE 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” (see page 355) 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 stanalone environment, it would distribute the jobs by itself.

If HPE 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 exits 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 HPE 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 384)
* LoadFactorDistModule, which distributes jobs across the cluster nodes based on the load factor. (See “LoadFactorDistModule” on page 364)
* QueueDistModule, which distributes jobs across the cluster nodes based on the number of currently running jobs on each cluster node. (See “QueueDistModule” on page 377)

# 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 (the 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 (Resource Manager), 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 72.
* Section “Job Counters” in the document *HPE Service Activator User’s and Administrator’s Guide*

Table 6‑1 ActivationModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| *min\_threads* | 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 |
| *max\_threads* | No | The maximum number of threads to allow for executing activations. | Yes | 20 |
| *queue\_class* | No | This can be set to the com.hp.ov.activator.mwfm. module.WeightedEngineQueue, com.hp.ov.activator.mwfm. module.SimpleEngineQueue, or com.hp.ov.activator.mwfm.module.PriorityEngineQueue  The WeightedEngineQueue uses the PRIORITY 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.  The SimpleEngineQueue will not do any prioritization of activation requests. They will be processed in FIFO order.  The PriorityEngineQueue uses the PRIORITY 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. | No | com.hp.ov. activator. mwfm. module.  WeightedEn  gineQueue |
| *refresh\_interva l* | No | The time interval between refreshing the internal cache which keeps data about which atomic tasks are deployed. | No | 60000  milliseconds |

Example: 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 related to workflow termination:

* 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 the file *$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 CLOB meaning that large amounts 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 CLOB)

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”.

Each Audit record is represented by an entry in the database table audit\_record, and eventually in audit\_record\_param as well. These tables are partitioned by the date field date\_time. The partition range can be defined in the module configuration. The module will then manage partitions automatically on both tables. Due to this, if Oracle database is being used, the database user must have enough privileges to manage partitions. This can be achieved by running the next SQL commands:

grant redefine any table to ${role};

grant EXECUTE ON DBMS\_REDEFINITION to ${role};

where ${role} must be replaced by the user role or the username. If the database user does not have enough privileges, any partition management operation will end up in the following error:

PLS-00201: identifier ‘DBMS\_REDEFINITION’ must be declared.

**Note**: The module never deletes partitions. This task, if desired, must be performed manually in the database.

It is granted by the module that partition ranges will be created based on meaningful periods of time. For example, if the configured partition interval is set to MONTHLY, each partition will end the first day of the next month. The exception here (in Oracle databases) is the partition interval BIWEEKLY, which will end up creating partitions every 15 days, no matter if the resulting date is meaningful or not.

Table 6‑2 AuditModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *exclude0*  *exclude1*  *...*  *excludeN* | 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 exclude parameters as you need | None |
| *store\_statistics* | No | Indicates whether to store or not workflow statistics. | Yes | “false” |
| *store\_serviceid\_and\_orderid* | No | If set to true, then ServiceId and OrderId parameters will be always stored no matter if the values are not provided. In that case, the values of these parameters will be set to the case-packet variables SERVICE\_ID and WORKFLOW\_ORDER\_ID.  ***This only applies to events triggered by the Audit node***. | Yes | “false” |
| *store\_audit* | No | Indicates whether to store or not audit records for all workflows. | Yes | “false” |
| *store\_userloginlogout* | No | Indicates whether to store or not audit records for login and logout events. | Yes | “true” |
| *store\_user\_roles* | No | If *store\_userloginlogout* is enabled, then it saves the roles for the login events. | Yes | “true” |
| *partition\_interval* | No | The time interval required for each partition that is going to be created by the module. Accepts the values BIWEEKLY, MONTHLY (default), BIMONTHLY, ONCE\_A\_QUARTER, ANNUAL, BIANNUAL and TRIANNUAL. | No | MONTHLY |

## BusinessCalendarModule

com.hp.ov.activator.mwfm.engine.module.umm.BusinessCalendarModule

The module enables workflows to work with the business calendars defined in HPE 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 be "business\_calendar\_module".

Example: 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>

## Cache Module

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 |
| *timeout* | Yes | Default timeout used when no value is provided by the node using this module. The value is specified in seconds. | No | None |
| *check\_interval* | Yes | Time, in seconds, between every check for expired content. | No | None |

Example: CacheModule

This example configures the CacheModule.

<Module>

<Name>cache\_mod</Name>

<Class-Name>

com.hp.ov.activator.mwfm.engine.module.CacheModule

</Class-Name>

<Param name="check\_interval" value="10"/>

<Param name="timeout" value="60"/>

</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 on the local host if this is set to true.

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 |
| --- | --- | --- | --- | --- |
| *dispatch\_local* | 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 |
| *casepacket* | Yes | The name of a case-packet variable which decides the distribution | Yes | None |
| *value0, value1 ... valueN* | No | The possible values for the parameter casepacket | Yes | None |
| *hostname0, hostname1 ... hostnameN* | No | The cluster nodes where the job can be distributed | Yes | None |
| *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: CasePacketDistModule

See the following configuration:

<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 HPE Service Activator is configured with more than one cluster node. The module can be started both when HPE Service Activator is started or as part of a reconfiguration. The module uses the master slave approch 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 |
| *check\_time\_poll\_interval* | No | Time interval in milliseconds at which this module will periodically ask each cluster node to give its system time | Yes. | 10000 |
| *allowable\_time\_delta* | No | Configurable value in milliseconds which is the allowable difference between the system times on master node and slave node. | Yes | 20000 |

Example: 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 controles 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 182.
* “MutexUnlock” on page 185.
* "MutexGetInfo” on page 181.
* "MutexSetInfo” on page 184.

Table 6‑6 ConflictModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *poll\_interval* | 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: 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 HPE 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 HPE 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 document *HPE Service Activator - System Integrator’s Overview*
* “User Management” in the document *HPE Service Activator - User’s and Administrator’s Guide*
* “Configuring Authentication or Authorization”, in the document *HPE Service Activator—Developing Plug-Ins and Compound Tasks*

Table 6‑7 DatabasedvancedAuthModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| *mwfm\_remote\_url* | 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 mwfm.xml file) the URL is:  //localhost:2000/wfm | No | None |
| sleep\_time | No | The time between the internal role cache is cleared. | No | 30 min |
| eight\_char\_password | No | If password should be trucated to 8 characters before authentication is done | No | false |
| secure\_user\_name | No | Transform the user name to a valid value, cutting the user name from the first invalid character. | No | false |
| password\_validation | No | true/false: specifies if password validation and expiration shall apply | No | false |
| 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 | The format checker class, where the complexity of the password is analyzed before accepting it.  HPE Service Activator provides a SimplePasswordFormatChecker which checks that the password is not equal to the user name and a ComplexPasswordFormatChecke r which also checks the password strength: usage of numbers and case letters.  Any other class may be defined here. If so, you can implement your password checker class just by extending the com.hp.ov.activator.mwfm. engine.module.umm.pwd.PasswordFormatChecker. | No | com.hp.ov.activat or.mwfm. module.u mm.Sim plePassw ordForm atChecker |
| allowed\_invalid\_login\_attepts | 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 |
| disable\_inactive\_users\_time | No | The number of days since the last time a user was successfully logged on until he has to be disabled. Defaults to 0, this meaning that no automatic disabling will be done. | No | 0 |

Example: 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 failurs 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 HPE 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 354
* “KeepAliveModule” on page 355

Example: DBTransactionModule

<Module>

<Name>transaction\_manager</Name>

<Class-Name>

com.hp.ov.activator.mwfm.engine.module.DBTransactionModule

</Class-Name>

</Module>

## DelegatedAuthModule

com.hp.ov.activator.mwfm.engine.module.umm.DelegatedAuthModule

This module supports delegated authentications based on SAML (Security Assertion Markup Language) v2 or OIDC (Open ID Connect).

This module delegates the authentication to a third-party service, managing the communication with the external service. In this sense, the user management is centralized in the external service, therefore, usernames or roles do not exist in HPE Service Activator. When a user logs in, the module sends a request to the external service which validates the request and returns an SAML assertion or JWT (JSON Web Token) with the result of the authentication, which includes relevant data, such as the expiration date, the roles, and other information (that can vary depending on the external service).

It supports several settings to adapt it for different customer installations. These are some of the available features in the module: to choose between SAML or OIDC, the SSO can be optionally enabled via Wildfly configuration, there are several parameters for configuring the HTTP client (that’s used by the module to communicate with the external IdP), the exchanged information could be encrypted and/or signed, the authentication API exposes a new method for validating a SAML assertion or JWT and so on.

### Architecture

The integration of delegated or federated authentication into a standard web application is a simple operation, mainly consisting of a set of changes in the webapp configuration. The login page is redirected to the external service login page which validates the data and returns an assertion/JWT to the application. This way, the web application cannot access the user credentials but trusts the external service. If users try to log in to another application (whose authentication is also managed by the same external service), they will be automatically logged in (without being prompted to provide credentials again), getting transparent SSO behavior.

However, HPE Service Activator cannot use this common approach because it contains several components that need authentication. For example, a third-party application can connect directly to the WFM engine using an RMI connection (or to the other interfaces: REST or SOAP). From the point of view of the integration with external authentication services, this includes an extra complexity that determines, for example, what profiles or bindings are used for the integration.

The module supports SAML v2 and OIDC. SAML is a set of standards for exchanging authentication and authorization data based on XML-based protocol. The technical definition of the standards has been developed by the OASIS organization. On the other hand, OIDC provides a similar solution, however, using different technology. OIDC is an identity layer on top of the OAuth 2 protocol that works with JSON Web Tokens (using HTTP communications).

#### SAML

SAML standards define a wide-range of aspects, for example, bindings, profiles, protocols, assertions, and schemas that provide support for almost any use case. Nevertheless, HPE Service Activator only implements those parts of the specification that support HPE Service Activator needs. The following is a detailed list of the selected elements:

* **Profile**: SAML specification supports several profiles (for example, Web Browser SSO, ECP, Identity Provider Discovery, Simple Logout) that are defined in the following specification document: <https://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf> This module implements the ECP profile (Enhanced Client or Proxy Profile), therefore, it acts like a proxy between the external auth service and the different components in HPE Service Activator, such as WFM, SOAP interface, REST interface, GUI.
* **Binding**: SAML specification supports several profiles (HTTP Redirect Binding, HTTP Artifact Binding, HTTP POST Binding) that are defined in the following specification document: <https://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf> This module implements the HTTP POST binding (which implies that messages are sent using an HTTP POST request to the external authentication service).
* **Protocol Messages**: SAML specification supports several options (for example, Assertion Query and Request, Authentication Request, Artifact Resolution, Name Identifier Managemen) that are defined in the following specification document: <https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf> This module uses the Authentication Request Protocol, implying that the module sends an AuthnRequest to the external service.
* **Schemas** - Authentication Context Classes: SAML specification supports several schemas to get the credentials and are defined in the following specification document: <https://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf> This module supports Password and PasswordProtectTransport (depending on whether the connection is secure or not).

This way, when users want to connect to the HPE Service Activator using RMI connections, they provide the credentials as part of the RMI instantiation. The RMI interface authenticates against the module which acts like an ECP and sends an AuthnRequest to the external auth service. When users are validated, the module processes the received assertion and confirms the validity and roles of the user to the RMI interface. If the same user tries to log in to the GUI, the module receives the request, checks if the assertion is still valid, and provides automatic access if it is. If the assertion has expired, then the module sends a new AuthnRequest to the external service.

Note that the Content-Type for the AuthnRequest is application/soap+xml.

#### OIDC

In case of the OIDC, the module works internally in a similar way, but the exchanged messages and the authorization flow are different. In this sense, OIDC supports several types of flows (for example, Authorization Code, Implicit, Client Credential) and grants (Authorization, Implicit and Hybrid).

This module uses the *Resource Owner Password Credentials Grant* (defined in the following specification: <https://tools.ietf.org/html/rfc6749#section-1.3.3> ), sending a JSON request with the following fields: username*,* password, grant\_type *(*equal to *password*), client\_id*,* scope *(*equal to *profile*), andclient\_secret. The Content-Type for the HTTP POST request is set to application/x-www-form-urlencoded.

The external service responds with an access token and a refresh token (which is used for refreshing the access token when the access token expires). The response is always verified using a symmetricKey (or an alternative key). If these keys are not configured, the response cannot be validated, and the module denies access.

#### Security

Delegated authentication means that service A (for example, HPE Service Activator) trusts service B (for example, a Keycloak service) to validate the credentials and to get the roles (authorization) for a specific user. This implies that relevant and secret information is going to be transported through the network and can be tampered. Therefore, it is important to make sure that the data is neither captured, nor modified externally.

SAML and OIDC provide several mechanisms to verify the authority and validity of the exchanged messages and it is important to understand how they work, adapting the module configuration for each use case based on the customer requirements, the supported features for the external Identity Provider, and so on.

The following are involved:

* Securing the channel: unless the communication between HPE SA servers and the external server for the authentication can be secured via a VPN or any other mechanism, it is highly recommended to secure connections, using HTTPS instead of HTTP. This can be configured via the configuration parameter https\_truestore\_file.
* Signing the SAML assertion
* Encrypting the SAML assertion
* Signing JWT
* Encrypting JWT

The signing of an SAML assertion or JWT ensures that the data has not been altered but it can be inspected if it is captured (using an HTTP connection). The encryption of the SAML assertion or JWT ensures that the data cannot be viewed without the key. Both signing and encrypting can be used separately and the use depends on the use case. For example, if a JWT is distributed to several services, then it is not necessary to encrypt it because the private key should be distributed to those services. On the other hand, if the JWT is only shared with a single service and contains critical data, then it is recommended to encrypt the token because the secret is only shared between two services.

**NOTE** For SAML authorization, the module sends the username and password in the request header (Base64 encoded), while for OIDC, the client\_secret can be sent as part of the request. It is highly recommended to force the use of HTTPS connections (even using self-signed certificates), avoiding the possibility that the communication is captured and this private data revealed.

#### Roles

The authentication information is completely managed in the external service, which implies that the required roles for HPE Service Activator must be provided in the SAML assertion or JWT token. This way, the roles should be created and assigned to users in the external service.

See Also

* The *Roles, Privileges, and Authentication* section of *HPE Service Activator - System Integrator’s Overview*
* The *Configuring Authentication or Authorization* section of *HPE Service Activator—Developing Plug-Ins and Compound Tasks*
* *Setting Roles* on page 28

For admin users, it is usually required to assign the roles ‘admin’ and ‘internal’.

Note that custom solutions can create its own roles that must be managed by the external service as well. For example, the roles for the built-in CRModelsolution are the following:

|  |  |
| --- | --- |
| Name | Description |
| WFManager | Role to enable the execution of Workflows which operate in CRModel tree |

The delegated authentication module does not support the superuser/team management feature.

However, it is possible to use the Role Mapping to simplify the integration with external services. If there is no role-mapping configuration, then the module will use all roles received from the external service. Otherwise, the module will remove those roles that are not in the mapping.

In addition to the standard role management, HPE Service Activator enables the association of an inventory tree view to a specific role. By default, users should not be able to get access to the inventory tree views in case the explicit associations do not exist. HPE Service Activator uses the internal roles (defined in the database) for this mapping, so if the external authentication service is not using the same role names, users can find problems with the access to the inventory tree. In general, it is required to create the same roles that are used by the external authentication service into the HPSA.

### Module configuration

Name this module “authenticator” in the configuration.

Table 6‑8 DelegatedAuthModule Parameters

| Parameter | Required | Description | Default |
| --- | --- | --- | --- |
| mode | Yes | Used to indicate the protocol: *SAML* or *OIDC* | None |
| url | Yes | URL for the external identity provider | None |
| mwfm\_remote\_url | Yes | It indicates the MWFM URL for internal operations. Normally it is set to ‘//localhost:2000/wfm’ | None |
| max\_connections | No | The maximum number of HTTP/HTTPS connections to the external service | No limit |
| max\_idle\_time | No | Milliseconds in idle state for HTTP/HTTPS connections before closing them | 1 minute |
| check\_expired\_sessions | No | Milliseconds for checking expired sessions by a dedicated thread (thread scheduling) | 1 minute |
| check\_http\_expired\_sessions | No | Milliseconds for checking expired HTTP connections before closing them (thread scheduling) | 10 seconds |
| clientid | Yes | Identifies the clientid in the external service | None |
| client\_secret | No | (Only for OIDC) If the external service requires a client secret for the authentication | None |
| expected\_audience | No | It defines a custom value to check against the “aud” information that is specified in the JWT or SAML assertion. If this parameter is not defined, the module will use the *clientid* as expected audience. | None |
| expiration\_time | No | The expiration time for sessions is configured externally. This property makes it possible to overwrite that value. | None |
| system\_user\_expiration\_time | No | Expiration time (in milliseconds) for the system user in milliseconds. If it is set to -1, the session does not expire (not recommended). | 1800000 |
| default\_expiration\_time | No | If the external service does not define the expiration time in the assertion/JWT, then this value is used as default expiration time instead (in milliseconds). | 300000 |
| saml\_role\_att\_id | No | (Only for SAML) A string that identifies the attribute name that contains the role elements. | Role |
| saml2\_auth\_schema | No | (Only for SAML) A string that defines the auth schema to use for the AuthnRequest. The module supports PasswordProtectedTransport (for HTTPS connections) and Password (for HTTP) connections. | urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport |
| logout\_url | No | A string to define a specific URL for the logout request. If not defined, the URL parameter is used. | None |
| saml\_consumer\_url | Yes, for SAML mode | A string that is used to set the AssertionConsumerServiceURL field in the AuthnRequest. This string must match the acceptable URL that is configured in the external service. |  |
| disable\_logout | No | It enables/disables the sending of logout requests to the external service. | false |
| expected\_issuer | No | A string used to compare against the issuer of the assertion/JWT. If it does not match, the response is considered invalid. If it is not defined, it does not get validated. | None |
| expected\_issuer2 | No | A string used to compare against the issuer of the assertion/JWT. If it does not match, the response is considered invalid. If it is not defined, it does not get validated. | None |
| expected\_issuer3 | No | A string used to compare against the issuer of the assertion/JWT. If it does not match, the response is considered invalid. If it is not defined, it does not get validated. | None |
| https\_truestore\_file | No | The complete filepath of the keystore that contains the certificate for the HTTPS – SSL connection. Note that it can be configured only if the *url* starts with https:// | None |
| https\_truestore\_pass\_file | No | The password for the keystore that contains the trusted certificate for the HTTPS – SSL connection | None |
| https\_truestore\_type | No | The type of the keystore | jks |
| signing\_pv\_key | No | (Only for SAML) A Base64 encoded string that contains the private key used to sign the sent messages (HPESA -> IdP) (internally managed into a PKCS8EncodedKeySpec). |  |
| signing\_pv\_algorithm\_key | No | (Only for SAML) The algorithm for the signing\_pv\_key | RSA |
| signing\_pv\_truestore\_file | No | (Only for SAML) It is possible to save the private key for signing messages in a keystore file. This is only used if the signing\_pv\_key is undefined. In this case, it contains the complete filepath for the keystore file. |  |
| signing\_pv\_truestore\_type | No | (Only for SAML) It defines the type of the keystore |  |
| signing\_pv\_truestore\_password | No | (Only for SAML) A string with the password of the keystore file |  |
| signing\_pv\_truestore\_key\_alias | No | (Only for SAML) A string with the alias of the private key for signing that is saved in the keystore |  |
| signing\_pv\_truestore\_key\_password | No | (Only for SAML) A string with the password for the private key |  |
| signing\_pb\_key | No | (Only for SAML) A Base64 encoded string with the public key that will be embedded into signed AuthnRequest messages as part of the KeyInfo element (internally managed as a PublicKey into a PKCS8EncodedKeySpec). This is used for the sent messages from HPESA -> IdP |  |
| signing\_pb\_algorithm\_key | No | (Only for SAML) The algorithm for the signing\_pb\_key |  |
| signing\_pb\_truestore\_file | No | (Only for SAML) A complete filepath for the keystore that contains the public key if the signing\_pb\_key is undefined |  |
| signing\_pb\_truestore\_type | No | (Only for SAML) The type of the keystore file |  |
| signing\_pb\_truestore\_password | No | (Only for SAML) The password to get access to the keystore |  |
| signing\_pb\_truestore\_key\_alias | No | (Only for SAML) The alias for the public key in the keystore |  |
| signing\_pb\_truestore\_key\_password | No | (Only for SAML) The password for the public key in the keystore |  |
| decrypt\_pv\_key | No | A string with the Base64 encoded private key used to decrypt the received SAML assertions |  |
| decrypt\_pv\_algorithm\_key | No | A string with the algoritm for the decrypt\_pv\_key | RSA |
| decrypt\_pv\_truestore\_file | No | The filepath for the keystore that contains the private key to decrypt the assertions. It is only used when the decrypt\_pv\_key is undefined. |  |
| decrypt\_pv\_truestore\_type | No | The type of keystore that contains the private key |  |
| decrypt\_pv\_truestore\_password | No | The password to get access to the keystore |  |
| decrypt\_pv\_truestore\_key\_alias | No | The alias for the private key into the keystore |  |
| decrypt\_pv\_truestore\_key\_password | No | The password for the private key |  |
| ext\_signing\_pb\_key | No | A string with the Base64-encoded public key that is used to verify the received token or SAML assertion |  |
| ext\_signing\_pb\_algorithm\_key | No | The algorithm for the ext\_signing\_pb\_key | RSA |
| ext\_signing\_pb\_truestore\_file | No | The filepath that contains the keystore that holds the key to verify the received JWT or SAML assertion. |  |
| ext\_signing\_pb\_truestore\_type | No | The keystore type |  |
| ext\_signing\_pb\_truestore\_password | No | The password to get access to the keystore |  |
| ext\_signing\_pb\_truestore\_key\_alias | No | The alias for the key in the keystore |  |
| ext\_signing\_pb\_truestore\_key\_password | No | The password for the key in the keystore |  |
| ext\_symmetric\_key | No | A string with the Symmetric Key used to verify the received JWT if it is defined. The module use this, and also the ext\_signing\_pb\_key/ext\_signing\_pb\_truestore\_file, if it is defined.  It normally holds the private key for the UOC, so HPESA can verify the UOC JWT. |  |
| disable\_signing\_check | No | The module will verify the signature for tokens and assertions, denying the access for the unsigned elements. In some scenarios, it could be interesting to accept non-signed elements. This boolean property allows to disable the validation. For SAML, the certificate for the signing validation uses to be embedded into the assertion but it can be read from an external source (configuring the origin using the properties: *ext\_signing\_pb\_truestore\_file* or alternatively *ext\_signing\_pb\_key* ). The module supports document and/or assertion signature. | false |
| http\_timeout | No | The connection timeout in milliseconds for the HTTP client that exchanges information with the external service. |  |
| http\_proxy | No | The URL for the proxy that will be set in the HTTP client. |  |
| saml\_nameid\_policy | No | (Only for SAML) A custom name-id for the composition of the AuthnRequest. | urn:oasis:names:tc:SAML:2.0:nameid-format:persistent |
| disable\_inactive\_users\_time | No | The number of days since the last time a user was successfully logged on until he has to be disabled. Defaults to 0, this meaning that no automatic disabling will be done. | 0 |

**NOTE** Make sure that the com.hp.ov.activator.session.HPSASessionListener has been configured in the $JBOSS\_HOME/standalone/deployments/hpsa.ear/activator.war/WEB-INF/web.xml file after running the ActivatorConfig tool. This listener will log out those users that have been logged in via the graphical user interface when the logout button is pressed, notifying to the IdP that the user has been logged out. In this sense the IdP will be aware about the session logout if it happens before the given expiration time.

The module does not support external logout actions managed by the IdP. The opened sessions should be closed manually by the users in HPE Service Activator, or automatically when they expire.

Note the module also supports the role mapping feature for associating rolenames (see more details in the section **Error! Reference source not found.**).

#### Internal user cache

Most of the services that rely in delegated authentication avoid to keep user data locally because it can cause conflicts in the synchronism between the local copy and the remote. Besides these services use to be characterized by many different users accessing concurrent to the same service what requires memory for caching data. Apart from that this approach doesn’t take advantage of the philosophy of the delegated authentication services.

The HPE Service Activator has not the same requirements as other applications like a social network due to these reasons:

* The number of users uses to be limited and small.
* The communication throught different internal layers requires short times for not impacting in the general performance.

Additionally to interact with an external system has an impact in the performance despite of using a pooled HTTP client.

In this way the DelegatedAuthModule keeps an internal cache for the logged in users. Once the user has been logged out or the user session has expired (because the SAML assertion or JWT token are not valid) the cache is cleaned. This has many benefits in terms of performance and also reduces the traffic between the IdP and the HPE Service Activator. But it has a side effect: the Service Activator won’t be aware about changes for users (like to disable the user, revoking permissions …) in the IdP until the session have expired locally and a new token will be requested.

This is particularly important if the module is configured with a custom expiration time for sessions (that overwrites the value that’s set in the SAML assertion/JWT).

Note there is a dedicated thread in the DelegatedAuthModule for checking expired sessions in the cache. It’s possible to configure the frequency of this check setting the desired time in the parameter check\_expired\_sessions (default value: 60000 ms)

### Module configuration examples

The module has been validated using Keycloak as external authentication service. The following examples are only some of the possible valid configurations:

OIDC + HTTPS:

<Module>

<Name>authenticator</Name>

<Class-Name>com.hp.ov.activator.mwfm.engine.module.umm.DelegatedAuthModule</Class-Name>

<Param name="mode" value="OIDC"/>

<Param name="mwfm\_remote\_url" value="//localhost:2000/wfm"></Param>

<Param name="url" value="https://localhost:8543/auth/realms/UOC/protocol/openid-connect/token"/>

<Param name="logout\_url" value="https://localhost:8543/auth/realms/UOC/protocol/openid-connect/logout"/>

<Param name="https\_truestore\_file" value="C:\HP\keycloak.jks"/>

<Param name="https\_truestore\_pass\_file" value="secret"/>

<Param name="clientid" value="HPESA"/>

<Param name="client\_secret" value="4300afde-8e82-4eb6-8f89-0aea1732df51"/>

<Param name="ext\_symmetric\_key" value="66X5Qu0="/>

</Module>

OIDC + HTTP:

<Module>

<Name>authenticator</Name>

<Class-Name>com.hp.ov.activator.mwfm.engine.module.umm.DelegatedAuthModule</Class-Name>

<Param name="mode" value="OIDC"/>

<Param name="mwfm\_remote\_url" value="//localhost:2000/wfm"></Param>

<Param name="url" value="https://localhost:8180/auth/realms/UOC/protocol/openid-connect/token"/>

<Param name="logout\_url" value="https://localhost:8180/auth/realms/UOC/protocol/openid-connect/logout"/>

<Param name="clientid" value="HPESA"/>

<Param name="client\_secret" value="4300afde-8e82-4eb6-8f89-0aea1732df51"/>

<Param name="ext\_symmetric\_key" value="66X5Qu0="/>

</Module>

SAML + encrypted assertions:

<Module>

<Name>authenticator</Name>

<Class-Name>com.hp.ov.activator.mwfm.engine.module.umm.DelegatedAuthModule</Class-Name>

<Param name="mode" value="SAML"/>

<Param name="mwfm\_remote\_url" value="//localhost:2000/wfm"></Param>

<Param name="url" value="http://localhost:8180/auth/realms/UOC/protocol/saml"/>

<Param name="saml\_consumer\_url" value=" http://localhost:8080/activator "/>

<Param name="clientid" value="mycustomclient"/>

<Param name="decrypt\_pv\_truestore\_file" value="C:\HP\keystore.jks"/>

<Param name="decrypt\_pv\_truestore\_password" value="secret"/>

<Param name="decrypt\_pv\_truestore\_privatekey\_alias" value="HPESA"/>

<Param name="decrypt\_pv\_truestore\_privatekey\_password" value="secret"/>

<Param name="expected\_issuer" value="http://localhost:8180/auth/realms/UOC"/>

</Module>

SAML:

<Module>

<Name>authenticator</Name>

<Class-Name>com.hp.ov.activator.mwfm.engine.module.umm.DelegatedAuthModule</Class-Name>

<Param name="mode" value="SAML"/>

<Param name="url" value="http://localhost:8180/auth/realms/UOC/protocol/saml"/>

<Param name="mwfm\_remote\_url" value="//localhost:2000/wfm"></Param>

<Param name="saml\_consumer\_url" value="http://localhost:8080/activator"/>

<Param name="clientid" value="HPESA"/>

<Param name="expected\_issuer" value="http://localhost:8180/auth/realms/UOC"/>

</Module>

### Wildfly configuration

The HPE Service Activator provides its custom login page where operators can fill the username and password. Sometimes customers need to get a full SSO integration and show the IdP login page instead of the default page. In this sense the operator won’t be prompted for the login data if he was already logged in other SSO company website. Note the configuration described in this section is optional and only necessary to configure SSO in Wildfly.

The SSO configuration relies on Wildfly and can be adapted via configuration files. Note that the configuration can vary depending on the technology, which can be either SAML or OIDC. In this sense the HPE Service Activator is not involved in the exchange of information between the IdP and the Keycloak Adapter in Wildfly, being a simple consumer from the Keycloak adapter. Consequently the available SSO functionality is restricted to the features offered by the Keycloak adapters. For further details, see the following:

* OIDC: <https://www.keycloak.org/docs/4.8/securing_apps/index.html#openid-connect-3>
* SAML: <https://www.keycloak.org/docs/4.8/securing_apps/index.html#saml-2>

The version of the Keycloak libraries is: 4.8.0

Although SAML and OIDC are technical standards (independent of custom implementations), the adapters have been developed to integrate Keycloak and Wildfly, therefore some conflicts with other IdP services are possible.

Required configuration to enable the SSO in the Service Activator:

1. Edit the file $JBOSS\_HOME/standalone/deployments/hpsa.ear/activator.war/WEB-INF/web.xml and replace the url-pattern in the security constraint as follows:

<security-constraint>

<web-resource-collection>

<web-resource-name>Restricted</web-resource-name>

**<url-pattern>/\*</url-pattern>**

</web-resource-collection>

<auth-constraint>

<role-name>HPSAUsers</role-name>

</auth-constraint>

<user-data-constraint>

<transport-guarantee>NONE</transport-guarantee>

</user-data-constraint>

</security-constraint>

1. Edit the $JBOSS\_HOME/standalone/deployments/hpsa.ear/activator.war/WEB-INF/web.xml file and change the login-config section.

* For OIDC:

<login-config>

<auth-method>KEYCLOAK</auth-method>

<realm-name>this is ignored currently</realm-name>

</login-config>

* For SAML:

<login-config>

<auth-method>KEYCLOAK-SAML</auth-method>

<realm-name>this is ignored currently</realm-name>

</login-config>

1. Edit the $JBOSS\_HOME/standalone/deployments/hpsa.ear/activator.war/WEB-INF/web.xml file and make sure that the new listener HPSASessionListener is configured:

<listener>

<listener-class>com.hp.ov.activator.session.HPSASessionListener</listener-class>

</listener>

1. Review the $JBOSS\_HOME/standalone/configuration/standalone.xml file and make sure that it contains the following two extensions:

…

<extension module="org.keycloak.keycloak-saml-adapter-subsystem"/>

<extension module="org.keycloak.keycloak-adapter-subsystem"/>

</extensions>

1. (Only for SAML) Edit the $JBOSS\_HOME/standalone/configuration/standalone.xml file and configure the following security domain:

<security-domain name="keycloak">

<authentication>

<login-module code="org.keycloak.adapters.jboss.KeycloakLoginModule" flag="required"/>

</authentication>

</security-domain>

1. (Only for SAML) Edit the $JBOSS\_HOME/standalone/deployments/hpsa.ear/META-INF/jboss-deployment-structure.xml file and add two third-party dependencies for the Wildfly Adapter:

…

<module name="io.undertow.servlet" services="export" />

<module name="io.undertow.core" services="export" />

…

1. Add the required security subsystem for the delegated authentication (nested in the profile section). The following are examples for SAML and OIDC, but note that you have to adjust the configuration parameters to your installation.

* For OIDC:

<subsystem xmlns="urn:jboss:domain:keycloak:1.1">

<secure-deployment name="activator.war">

<realm>UOC</realm>

<resource>hpsaoidcclient</resource>

<auth-server-url>http://localhost:8180/auth</auth-server-url>

<ssl-required>external</ssl-required>

<credential name="secret">4300afde-8e82-4eb6-8f89-0aea1723df51</credential>

</secure-deployment>

</subsystem>

* For SAML:

<subsystem xmlns="urn:jboss:domain:keycloak-saml:1.1">

<secure-deployment name="activator.war">

<SP entityID="hpesp\_saml" sslPolicy="NONE" nameIDPolicyFormat="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" logoutPage="http://localhost:8081/activator/jsp/logout.jsp">

<IDP entityID="idp">

<SingleSignOnService signRequest="false" validateResponseSignature="false" validateAssertionSignature="false" requestBinding="POST" bindingUrl="http://pompeiv5.gre.hpecorp.net:8180/auth/realms/HPESP/protocol/saml" assertionConsumerServiceUrl="http://localhost:8081/activator/saml"/>

<SingleLogoutService validateRequestSignature="false" validateResponseSignature="false" signRequest="false" signResponse="false" requestBinding="POST" responseBinding="REDIRECT" postBindingUrl="http://pompeiv5.gre.hpecorp.net:8180/auth/realms/HPESP/protocol/saml" redirectBindingUrl="http://localhost:8081/activator/jsp/logout.jsp"/>

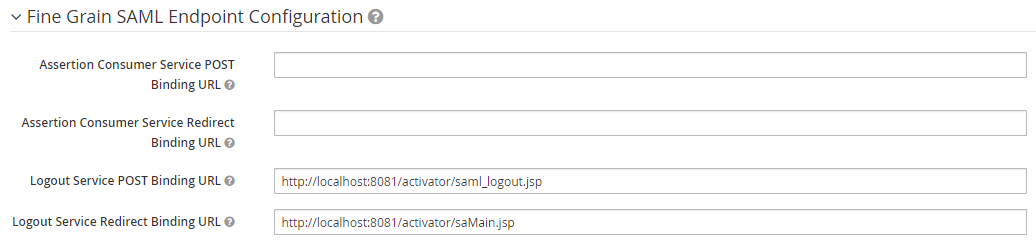
</IDP>

</SP>

</secure-deployment>

</subsystem>

The IdP configuration for the previous SAML snippet has to contain the binding URLs. Example:



For further details, see the official documentation: <https://www.keycloak.org/docs/4.8/securing_apps/index.html#securing-wars-via-adapter-subsystem>

### Internal Authentication API

The DelegatedAuthModule is targeted for those environments where the authentication and authorization are handled by an external service (commonly named IdentityProvider). This is particularly important to understand what possibilities are available in the com.hp.ov.activator.mwfm.WFAuthenticator API that exposes the following two main methods:

* WFManager login(String username, String password) throws RemoteException, AuthException;
* WFManager loginIDP(String token, String refreshToken, String username) throws RemoteException, AuthException;

There are other methods that vary slightly and whose reference can be found in the javadoc.

The login method let users get a WFManager instance using credentials that is validated by the Identity Provider. The DelegatedAuthModule sends a request to the IdP and processes the response to extract information like roles, session expiration time, and so on.

Note that this is not the most common use of the delegated authentication, where the login pages are redirected to the Identity Provider login page, where users can fulfill the credentials and to be redirected back to the application. It avoids the sharing of sensible information with third parties and also centralizes the user management.

Although HPE Service Activator supports it via the Wildfly configuration, it is not used by the DelegatedAuthModule for the internal API. In this particular case, the module acts a client proxy, getting the credentials from users and sending a request to the IdP that will validate them. Once the request has been validated, the module returns a WFManager instance. This approach is used by components without a GUI, like the REST and SOAP interfaces.

However, the API also offers an additional loginIDP method to be used by other HPE applications in the activation ecosystem. This method does not interact with an external Identity Provider service but validates the data of the token to ensure the authenticity. It accepts three input parameters:

* String token: contains a JWT token (formatted according to the IETF RFC 759) or a SAML assertion (encoded in a Base64 format).
* String refreshToken: contains a JWT refresh token when the main token had expired (it applies only for OIDC) and is mainly used internally. Set to null in those cases where users do not have a refresh token.
* String username: the username associated to the token. It’s usually extracted from the token but could be overwriten using this parameter.

The module extracts the roles, session expiration time, and so on, from the token, so it is important to ensure the content has not been tampered through the signature verification. The module supports several configuration parameters for checking the signature and to decrypt it if it is necessary. Generally, the JWT/SAML assertions are signed, but note that it is also recommended to encrypt them (specially for non secured transport layers).

The following paragraphs explain how the content of the token is processed by the module. Note that the content can vary depending on the IdentityProvider, so it is important to identify what information is used and how.

For SAML, the following fields are processed:

* Signature: its content is extracted from the next XPath /Response/Signature. The module uses the embedded key in the signature to verify if. The module let configure the public if the key is not embedded or if administrators want to enforce the use of an external key through the following parameters:
  + ext\_signing\_pb\_key and ext\_signing\_pb\_algorithm\_key to set the public key content as parameter.
  + ext\_signing\_pb\_truestore\_file, ext\_signing\_pb\_truestore\_type, ext\_signing\_pb\_truestore\_password, ext\_signing\_pb\_truestore\_key\_alias and ext\_signing\_pb\_truestore\_key\_password to define the keystore containing the key.

The signature validation can be disabled by setting disable\_signing\_check parameter to false.

* Recipient: its content is extracted from /Response/Subject/SubjectConfirmation/SubjectConfirmationData/@Recipient and must match the value of the saml\_consumer\_url parameter.
* Username: its content is extracted from /Response/Subject/NameID
* Expiration time: SAML supports different expiration times in different sections of the assertion, each one oriented for a different purpose. In our particular case, the module validates all the AuthnStatements present in the assertion, ensuring that the /Response/AuthnStatement/@SessionNotOnOrAfter are lower or equal than the current moment. Normally, different AuthnStatements contain the same expiration time, so the module uses the last one to use it as internal expiration time. This check can be disabled by setting the disable\_check\_expiration\_time parameter to false. The expiration time can be overwritten also using the expiration\_time parameter.
* Audience: defines who requested the assertion and the value is extracted from /Response/Conditions/AudienceRestriction/Audience and validated against the value of the expected\_audience parameter. If this parameter is not set, then the mandatory clientid parameter is used by default.
* Issuer: defines who generated the assertion and the value is extracted from /Response/Issuer and validated against the value of the expected\_issuer parameter.
* Status: defines the status of the SAML assertion and the value is extracted from /Response/Status/StatusCode/@Value and must match the following text: “urn:oasis:names:tc:SAML:2.0:status:Success”.
* Roles: are extracted from the attributes that match the xpath /Response/AttributeStatement/Attribute whose name (defined in the attribute: /Response/AttributeStatement/Attribute/@Name) is equal to the text “Role”. It is possible to instruct the module to use another text through the saml\_role\_att\_id parameter.

For OIDC, the following fields of the JWT are processed:

* Signature: is usually validated through a symmetric key (whose value is read from the ext\_symmetric\_key parameter and decrypted in the module with the HPE Service Activator crypt tool), or alternatively setting the key through:
  + Directly in the ext\_signing\_pb\_key and ext\_signing\_pb\_algorithm\_key parameters
  + Using a keystore: ext\_signing\_pb\_truestore\_file, ext\_signing\_pb\_truestore\_type, ext\_signing\_pb\_truestore\_password, ext\_signing\_pb\_truestore\_key\_alias, or ext\_signing\_pb\_truestore\_key\_password

Note that the validation can be disabled setting the disable\_signing\_check the parameter to false.

* Audience: is defined in the /aud and its value must match the value of the expected\_audience parameter. Note that JWT can contain an array of audiences where at least one of them must match with the configured one.
* Roles: are extracted from:
  + /realm\_access/roles
  + /roles
  + /resource\_access/$expected\_audience$/roles (where the expected\_audience is defined in the expected\_audience parameter).

Note that the roles can be mapped to an internal roles using the rolemapping feature available in HPE Service Activator.

* + Expiration time: is defined in the optional element /exp . The module enables the configuration of a default expiration time (using the default\_expiration\_time parameter) in case of an undefined value. Additionally, it is possible to force-set a custom value using the expiration\_time parameter.
* Issuer: is defined in the element /iss and the value must match with the value of one of the following parameters: expected\_issuer, expected\_issuer2, expected\_issuer3
* IssueAt: is defined in the element /iat and defines when the JWT was composed. The module checks this time to ensure the time won’t be greater than the current moment.
* Id: is defined in the /jti element and contains a unique identifier for the JWT.
* Username: is defined in the custom /preferred\_username element, being a mandatory field for the module.

Generally, JWT tokens or SAML assertions are signed but they could be also encrypted. In this sense, it is possible to instruct the module for decrypting it:

* Setting the private key directly in the decrypt\_pv\_key and decrypt\_pv\_algorithm\_key parameters.
* Or using a keystore: decrypt\_pv\_truestore\_file, decrypt\_pv\_truestore\_type, decrypt\_pv\_truestore\_password, decrypt\_pv\_truestore\_key\_alias, or decrypt\_pv\_truestore\_key\_password

### IdP Requests

Depending of the configured mode (SAML or OIDC) the module will initiate the communication with the IdP in a different way.

#### SAML

In this mode the module acts as an proxy ( EnhancedClientProxy or ECP ) being an intermediarie between the Service Provider and the Identity Provider (see more details about the ECP in the Oasis specification: <http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html> , section: 5.2). In the Service Activator context, the different components, like the SOAP interface, REST interface, engine … are considered Service Providers while the module is the proxy between these components and the IdP.

To reduce the impact in the different components, they continue passing the credentials to the module through the internal API and the module will encapsulate and manage the communication with the IdP.

Once a component requests to authenticate an user, the module will generate an AuthnRequest for login operations and a logout request to inform the IdP that user has been logged out before the expiration.

Note next requests are using the encoding: UTF-8.

##### Login

The characteristics of the AuthnRequest are:

* Protocol: urn:oasis:names:tc:SAML:2.0:protocol
* Binding: urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST
* Policy: urn:oasis:names:tc:SAML:2.0:nameid-format:persistent (although it can be overwriten using the module configuration parameter: saml\_nameid\_policy).
* Issuer: defined in the module configuration parameter: clientid
* ConsumerURL: defined in the module configuration parameter: saml\_consumer\_url
* IssueInstant: current moment.
* Version: VERSION\_20

The request is signed if the key is configured in the module using one of the next set of properties:

* Key specified in the parameter: signing\_pv\_key and signing\_pv\_algorithm\_key.
* Using a keystore: signing\_pv\_truestore\_file, signing\_pv\_truestore\_type, signing\_pv\_truestore\_password, signing\_pv\_truestore\_key\_alias and signing\_pv\_truestore\_key\_password.

If it’s possible to embed the public key in the request so Identity Providers can use it to verify the signature. The public key has to be defined using one of the next set of properties:

* Key specified in the parameter: signing\_pb\_key and signing\_pb\_algorithm\_key.
* Using a keystore: signing\_pb\_truestore\_file, signing\_pb\_truestore\_type, signing\_pb\_truestore\_password, signing\_pb\_truestore\_key\_alias and signing\_pb\_truestore\_key\_password.

Once the XML typed content of the request is generated, the module will send a HTTP Post request to the identity provider with the next headers:

* Content-Type: application/soap+xml
* Authorization: Basic + a Base64 enconded text for the pair: username:password

##### Logout

The logout request is not standardized but it’s required for some services like Keycloak. It’s possible to disable it setting the module configuration parameter disable\_logout to false.

The content of the logout request is characterized by:

* Version: VERSION\_20
* Issuer: defined in the parameter clientid
* Destination: the URL for the logout endpoint
* IssueInstant: current moment
* Issue NotOnOrAfter: valid for 5 minutes after the current momentl.
* Logout reason: “HPE Service Activator Logout”
* SessionIndex: the session identifier returned in the response for the AuthnRequest.

It can be signed (and the public key embedded in the request) using the same properties that are defined for the Login operation. The message is transported using a HTTP Post with the same format than the Login.

#### OIDC

In the OIDC case the requests are more simple, and the length of the trasmmited data short. In general it is a modern standard easy to integrate with third party services.

##### Login

The HTTP Post request contains this header:

* Content-Type : application/x-www-form-urlencoded

And the URL contains next encoded parameters:

* username: the name of the user to be authenticated
* password
* grant\_type: a constant “password”
* client\_id: defined in the module configuration parameter clientid
* scope: a constant “profile”
* client\_secret: defined in the module configuration parameter client\_secret (optional).

##### Refresh token

Generally JWTs have to be renew (or refreshed) every certain time, ensuring the user is still valid for the service provider. The HTTP Post request contains this header:

* Content-Type : application/x-www-form-urlencoded

And the URL contains next encoded parameters:

* refresh\_token: the refresh token obtained from the last response of the IdP.
* grant\_type: a constant “refresh\_token”
* client\_id: defined in the module configuration parameter clientid
* scope: a constant “profile”
* client\_secret: defined in the module configuration parameter client\_secret (optional).

##### Logout

Once the user has requested a HTTP Post request is sent to the IdP with this header:

* Content-Type : application/x-www-form-urlencoded

And the URL contains next encoded parameters:

* username: the name of the user to be authenticated
* client\_id: defined in the module configuration parameter clientid
* client\_secret: defined in the module configuration parameter client\_secret (optional).
* user\_id: internal user identifier in the IdP that’s extracted from the received JWT.
* token: last received JWT token.
* refresh\_token: last received JWT refresh token.

**NOTE** The previous requests don’t manage encrypted credentials. It’s strongly recommended to use SSL for communicating with the IdentityProvider. The section **Error! Reference source not found.** explains how to configure the module for using HTTPS.

### Token examples

The following are some examples that have been extracted from a testing environment:

* JWT: eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJwMlBTbmduMHhvbjRLc2VTcTduaWxRMjJUNVpqX0wxN3VpNDBiU3FYZ2F3In0..Bxv9Gtby2GyaFG2de1HwU7Vto7nLA7yIE34kNl-jye0hraAF488dCbO\_5dCWfixMQ\_\_38wkjlW7BdP8qRzR6SdrDTsCB9tZwfnofpKpG03bOT4fOBKCXNQ4o3d\_Xuc6eTZmeGuBVL4v2MGP5TOj9LjzCNuTQZVViDhnsWM5zfMdQLNQpk2BpbnCFE-6wuZN8uN4HWGF0ux3O1DmXY5sVeB2HV5v1LwNkY2dnWf9HB\_mxCmRju3m3KYcFxZbG3ymTuiidRjLS8cznWYmPrB67E35O2uquLsrDGOYWmnZHkTVvxNPFtNiTFQpP43ep5-nFxphCiS8WHmaw8Lpe8DgUiA

Where the corresponding decoded parts (header, payload and signature) are:

* + Header:

{

  "alg": "RS256",

  "typ": "JWT",

  "kid": "p2PSngn0xon4KseSq7nilQ22T5Zj\_L17ui40bSqXgaw"

}

* + Payload:

{

  "jti": "64df380f-d933-4bf3-9b65-503ae9599f69",

  "exp": 1540809615,

  "nbf": 0,

  "iat": 1540809315,

  "iss": "<http://127.0.0.1:8180/auth/realms/UOC>",

  "aud": "mycustomclient2",

  "sub": "5e287f85-3c09-4132-86f6-8c6059578b3f",

  "typ": "Bearer",

  "azp": "mycustomclient2",

  "auth\_time": 0,

  "session\_state": "1cf6c70f-4b93-449b-aac3-44220dd71b99",

  "acr": "1",

  "allowed-origins": [],

  "realm\_access": {

    "roles": [

      "User Administrator",

      "offline\_access",

      "uma\_authorization"

    ]

  },

  "resource\_access": {

    "UOC": {

      "roles": [

        "User Administrator",

        "Package Designer",

        "Guest",

        "Operator\_L3",

        "Platform Administrator",

        "Operator\_L1",

        "Operator\_L2",

        "Report\_Exporter"

      ]

    },

    "account": {

      "roles": [

        "manage-account",

        "manage-account-links",

        "view-profile"

      ]

    }

  },

  "scope": "profile mycustomclient2 email",

  "email\_verified": false,

  "preferred\_username": "admin",

  "email": "[admin@hpe.com](mailto:admin@hpe.com)"

}

* + Signature: --
* SAML (after to decode the Base64 text):

<SOAP-ENV:Envelope xmlns:SOAP-ENV="<http://schemas.xmlsoap.org/soap/envelope/>" xmlns:ecp="urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp" xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">

  <SOAP-ENV:Header>

    <ecp:Response AssertionConsumerServiceURL="<http://localhost:3000/workspace-manager/ECP>" SOAP-ENV:actor="<http://schemas.xmlsoap.org/soap/actor/next>" SOAP-ENV:mustUnderstand="1"/>

  </SOAP-ENV:Header>

  <SOAP-ENV:Body>

    <samlp:Response Destination="<http://localhost:3000/workspace-manager/ECP>" ID="ID\_bb5da1d2-b76d-4760-b67f-e75ea0586bc1" InResponseTo="\_b2c92bc8216ff317a5c42d797a7cd8ca" IssueInstant="2018-10-25T07:17:12.742Z" Version="2.0">

      <saml:Issuer>[http://localhost:8180/auth/realms/UOC</saml:Issuer](http://localhost:8180/auth/realms/UOC%3c/saml:Issuer)>

      <dsig:Signature xmlns:dsig="[http://www.w3.org/2000/09/xmldsig#](http://www.w3.org/2000/09/xmldsig)">

        <dsig:SignedInfo>

          <dsig:CanonicalizationMethod Algorithm="[http://www.w3.org/2001/10/xml-exc-c14n#](http://www.w3.org/2001/10/xml-exc-c14n)"/>

          <dsig:SignatureMethod Algorithm="<http://www.w3.org/2001/04/xmldsig-more#rsa-sha256>"/>

          <dsig:Reference URI="#ID\_bb5da1d2-b76d-4760-b67f-e75ea0586bc1">

            <dsig:Transforms>

              <dsig:Transform Algorithm="<http://www.w3.org/2000/09/xmldsig#enveloped-signature>"/>

              <dsig:Transform Algorithm="[http://www.w3.org/2001/10/xml-exc-c14n#](http://www.w3.org/2001/10/xml-exc-c14n)"/>

            </dsig:Transforms>

            <dsig:DigestMethod Algorithm="<http://www.w3.org/2001/04/xmlenc#sha256>"/>

            <dsig:DigestValue>S815U+Agf9TYGSctqJkdtxRtIOi9kq4POw5MsEuhz4g=</dsig:DigestValue>

          </dsig:Reference>

        </dsig:SignedInfo>

        <dsig:SignatureValue>E0CNNsiS9rwMR6tRWnApmf5SKY9BbKGYPgCt7vek30CBn/IQXmLw+A43nYHTpcBNjKoTmnAQf0/N&#xD;

uiIV7o5u+GH2zgw0WbUVtoYiJeb9PlbaG6XW7a3hkW6sVUeCHyLRgYwJYUA2/Y6ehoiBiAq4gg0M&#xD;

p+IMDv9t/MdKgAy+xgXdBCfVrAkI+UyRlzBZfBdlvK/QA8jbp/kkd8J+RC5fPxDR2RelcpMPgM9b&#xD;

EpPMMhCZVoHAJu8xgTC3808b9pEYxy0KaOQCj/C9e6wJgM9cQivkQPJxrc5aBaGyqOMEAHoxeqAB&#xD;

T3atqC5tIkX4mnQqGqAC2UdS4sQtIcKC8oUHQA==</dsig:SignatureValue>

        <dsig:KeyInfo>

          <dsig:KeyName>p2PSngn0xon4KseSq7nilQ22T5Zj\_L17ui40bSqXgaw</dsig:KeyName>

          <dsig:X509Data>

            <dsig:X509Certificate>MIIClTCCAX0CBgFmoWkXwjANBgkqhkiG9w0BAQsFADAOMQwwCgYDVQQDDANVT0MwHhcNMTgxMDIz&#xD;

MTQ1MDExWhcNMjgxMDIzMTQ1MTUxWjAOMQwwCgYDVQQDDANVT0MwggEiMA0GCSqGSIb3DQEBAQUA&#xD;

A4IBDwAwggEKAoIBAQCFPv5IlcsBTkYeC9BWFalnvBgIweW71Nhi6orWbJSLDey5UPIHb51xafjH&#xD;

1oBuhcc//jsTxtwx00Wsk+c2ARjk462NKBgHqdX9CLe57sekQwNVRPvAatmjRG33IhB08mXk/Ns2&#xD;

KRw6EFzwiVg8jPc42kq2AFP7M+Ve66OwKUcPVPha0kY9DRGsS8vmSH0bMNaYF3r8dvu3LgnU8mdx&#xD;

+/rkeAqB5e0kB0aDSt0Z0ASVVRFFej4objFNeeotZyOVo5Mo7F+f0/44vrexrxqRUP68dd5RBlb8&#xD;

VUu2DwO7HidND7W6j4bRKJgjQ5wp3krAcexPHfDmAAIsMk8F8AwmS5fxAgMBAAEwDQYJKoZIhvcN&#xD;

AQELBQADggEBAF8oWZ9U9+heebPUAhGYqQNRk1TohIHJaFGD0ho2zgLo5MKvUCWYmaYcsI7MRM/P&#xD;

3P77Tp6iKqpjz9N0VtijLaCSZIgyj1h7mWhyALxD90oXIQUPz/bDlXYTMI9+XEkecnaVWdU6GixC&#xD;

1TxWo2yyzJIvJxR7uHiWvhT8ORltGK5Z/01ZLTbX/x0/xoPeTFMxZMPhtMAMzNVZOIatKZc8weIT&#xD;

MVlxFVlw1iLOA8IgHScvQLkoMbEokN9/JN9KC3EPWCG7NHnyxQNfJYy8Oxj7Qrxig58QdU90mpcc&#xD;

38jEdcbTqu+nHVWsenyXh+PktLTGhjQCeJDdNpcwep4/66X5Qu0=</dsig:X509Certificate>

          </dsig:X509Data>

          <dsig:KeyValue>

            <dsig:RSAKeyValue>

              <dsig:Modulus>hT7+SJXLAU5GHgvQVhWpZ7wYCMHlu9TYYuqK1myUiw3suVDyB2+dcWn4x9aAboXHP/47E8bcMdNF&#xD;

rJPnNgEY5OOtjSgYB6nV/Qi3ue7HpEMDVUT7wGrZo0Rt9yIQdPJl5PzbNikcOhBc8IlYPIz3ONpK&#xD;

tgBT+zPlXuujsClHD1T4WtJGPQ0RrEvL5kh9GzDWmBd6/Hb7ty4J1PJncfv65HgKgeXtJAdGg0rd&#xD;

GdAElVURRXo+KG4xTXnqLWcjlaOTKOxfn9P+OL63sa8akVD+vHXeUQZW/FVLtg8Dux4nTQ+1uo+G&#xD;

0SiYI0OcKd5KwHHsTx3w5gACLDJPBfAMJkuX8Q==</dsig:Modulus>

              <dsig:Exponent>AQAB</dsig:Exponent>

            </dsig:RSAKeyValue>

          </dsig:KeyValue>

        </dsig:KeyInfo>

      </dsig:Signature>

      <samlp:Status>

        <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>

      </samlp:Status>

      <saml:Assertion xmlns="urn:oasis:names:tc:SAML:2.0:assertion" ID="ID\_265d187e-bcb6-41d9-9662-02a4e78e613e" IssueInstant="2018-10-25T07:17:12.742Z" Version="2.0">

        <saml:Issuer>[http://localhost:8180/auth/realms/UOC</saml:Issuer](http://localhost:8180/auth/realms/UOC%3c/saml:Issuer)>

        <dsig:Signature xmlns:dsig="[http://www.w3.org/2000/09/xmldsig#](http://www.w3.org/2000/09/xmldsig)">

          <dsig:SignedInfo>

            <dsig:CanonicalizationMethod Algorithm="[http://www.w3.org/2001/10/xml-exc-c14n#](http://www.w3.org/2001/10/xml-exc-c14n)"/>

            <dsig:SignatureMethod Algorithm="<http://www.w3.org/2001/04/xmldsig-more#rsa-sha256>"/>

            <dsig:Reference URI="#ID\_265d187e-bcb6-41d9-9662-02a4e78e613e">

              <dsig:Transforms>

                <dsig:Transform Algorithm="<http://www.w3.org/2000/09/xmldsig#enveloped-signature>"/>

                <dsig:Transform Algorithm="[http://www.w3.org/2001/10/xml-exc-c14n#](http://www.w3.org/2001/10/xml-exc-c14n)"/>

              </dsig:Transforms>

              <dsig:DigestMethod Algorithm="<http://www.w3.org/2001/04/xmlenc#sha256>"/>

              <dsig:DigestValue>5pRDiO3TMOZd+Trmwct/CSkOZ9Yr0YPrA0OXPDbQTOc=</dsig:DigestValue>

            </dsig:Reference>

          </dsig:SignedInfo>

          <dsig:SignatureValue>XI2BsrFANX+/ydCnYxHXd/FTIHdeA1QZHYOyo5bLZeFcXumSpU/ekmFHtlZOG1phLKEL0lXeaIxG&#xD;

eAlGvGi6N9pBs9s1p+YT/Y7zqphJHCvuEAeTsyfUI3ht7S3mm1ZF1hX7bFNwrdyvkZOZ2igYhCON&#xD;

3Tl6vTldNVUWAypVW79iGRgSG3X/yzo39VTEnNvc7jFrrHP0eaXPBgfuRzgjdu5C56H5e0V4PH/d&#xD;

PkQwEe6A7H6fJdN9huFPKCXoJmvc4mQemTLbrvolJ9PuvaEDD2z+TlHQ6mnSfM7znwgZUkTAQlBh&#xD;

3cIdGjHdHD+R5rf2qJY9x+ZTVPoaAe0jxyTzpw==</dsig:SignatureValue>

          <dsig:KeyInfo>

            <dsig:KeyName>p2PSngn0xon4KseSq7nilQ22T5Zj\_L17ui40bSqXgaw</dsig:KeyName>

            <dsig:X509Data>

              <dsig:X509Certificate>MIIClTCCAX0CBgFmoWkXwjANBgkqhkiG9w0BAQsFADAOMQwwCgYDVQQDDANVT0MwHhcNMTgxMDIz&#xD;

MTQ1MDExWhcNMjgxMDIzMTQ1MTUxWjAOMQwwCgYDVQQDDANVT0MwggEiMA0GCSqGSIb3DQEBAQUA&#xD;

A4IBDwAwggEKAoIBAQCFPv5IlcsBTkYeC9BWFalnvBgIweW71Nhi6orWbJSLDey5UPIHb51xafjH&#xD;

1oBuhcc//jsTxtwx00Wsk+c2ARjk462NKBgHqdX9CLe57sekQwNVRPvAatmjRG33IhB08mXk/Ns2&#xD;

KRw6EFzwiVg8jPc42kq2AFP7M+Ve66OwKUcPVPha0kY9DRGsS8vmSH0bMNaYF3r8dvu3LgnU8mdx&#xD;

+/rkeAqB5e0kB0aDSt0Z0ASVVRFFej4objFNeeotZyOVo5Mo7F+f0/44vrexrxqRUP68dd5RBlb8&#xD;

VUu2DwO7HidND7W6j4bRKJgjQ5wp3krAcexPHfDmAAIsMk8F8AwmS5fxAgMBAAEwDQYJKoZIhvcN&#xD;

AQELBQADggEBAF8oWZ9U9+heebPUAhGYqQNRk1TohIHJaFGD0ho2zgLo5MKvUCWYmaYcsI7MRM/P&#xD;

3P77Tp6iKqpjz9N0VtijLaCSZIgyj1h7mWhyALxD90oXIQUPz/bDlXYTMI9+XEkecnaVWdU6GixC&#xD;

1TxWo2yyzJIvJxR7uHiWvhT8ORltGK5Z/01ZLTbX/x0/xoPeTFMxZMPhtMAMzNVZOIatKZc8weIT&#xD;

MVlxFVlw1iLOA8IgHScvQLkoMbEokN9/JN9KC3EPWCG7NHnyxQNfJYy8Oxj7Qrxig58QdU90mpcc&#xD;

38jEdcbTqu+nHVWsenyXh+PktLTGhjQCeJDdNpcwep4/66X5Qu0=</dsig:X509Certificate>

            </dsig:X509Data>

            <dsig:KeyValue>

              <dsig:RSAKeyValue>

                <dsig:Modulus>hT7+SJXLAU5GHgvQVhWpZ7wYCMHlu9TYYuqK1myUiw3suVDyB2+dcWn4x9aAboXHP/47E8bcMdNF&#xD;

rJPnNgEY5OOtjSgYB6nV/Qi3ue7HpEMDVUT7wGrZo0Rt9yIQdPJl5PzbNikcOhBc8IlYPIz3ONpK&#xD;

tgBT+zPlXuujsClHD1T4WtJGPQ0RrEvL5kh9GzDWmBd6/Hb7ty4J1PJncfv65HgKgeXtJAdGg0rd&#xD;

GdAElVURRXo+KG4xTXnqLWcjlaOTKOxfn9P+OL63sa8akVD+vHXeUQZW/FVLtg8Dux4nTQ+1uo+G&#xD;

0SiYI0OcKd5KwHHsTx3w5gACLDJPBfAMJkuX8Q==</dsig:Modulus>

                <dsig:Exponent>AQAB</dsig:Exponent>

              </dsig:RSAKeyValue>

            </dsig:KeyValue>

          </dsig:KeyInfo>

        </dsig:Signature>

        <saml:Subject>

          <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">admin</saml:NameID>

          <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">

            <saml:SubjectConfirmationData InResponseTo="\_b2c92bc8216ff317a5c42d797a7cd8ca" NotOnOrAfter="2018-10-25T07:22:10.742Z" Recipient="<http://localhost:3000/workspace-manager/ECP>"/>

          </saml:SubjectConfirmation>

        </saml:Subject>

        <saml:Conditions NotBefore="2018-10-25T07:17:10.742Z" NotOnOrAfter="2018-10-25T07:18:10.742Z">

          <saml:AudienceRestriction>

            <saml:Audience>mycustomclient</saml:Audience>

          </saml:AudienceRestriction>

        </saml:Conditions>

        <saml:AuthnStatement AuthnInstant="2018-10-25T07:17:12.742Z" SessionIndex="e8ca43e7-c58a-48e9-9cf2-5449d236b48d::34268585-cf4c-4ca0-a43a-27c6477cfed5">

          <saml:AuthnContext>

            <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified</saml:AuthnContextClassRef>

          </saml:AuthnContext>

        </saml:AuthnStatement>

        <saml:AttributeStatement>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">view-profile</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">Operator\_L1</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">uma\_authorization</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">manage-account</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">manage-account-links</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">Guest</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">Package Designer</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">User Administrator</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">offline\_access</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">Report\_Exporter</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">User Administrator</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">Operator\_L3</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">Operator\_L2</saml:AttributeValue>

          </saml:Attribute>

          <saml:Attribute Name="Role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">

            <saml:AttributeValue xmlns:xs="<http://www.w3.org/2001/XMLSchema>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:type="xs:string">Platform Administrator</saml:AttributeValue>

          </saml:Attribute>

        </saml:AttributeStatement>

      </saml:Assertion>

    </samlp:Response>

  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>

## HTTPRequestModule

com.hp.ov.activator.mwfm.engine.module.monitor.HTTPRequestModule

The HTTPRequestModule 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 HTTPRequestModule; 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 HTTPRequestModule 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 HTTPRequestModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| *host* | Yes | The target URL for the HTTP(S) connection | No | None |
| *keystore* | No | The location of the SSL keystore file necessary for HTTPS client authentication | No | 30 min |
| *keypass* | No | The password for the public certificate/private key pair, necessary for HTTPS client authentication | No | None |
| *storepass* | No | The password to access the keystore file, necessary for HTTPS | No | None |
| *username* | No | Username for network connection authentication | No | None |
| *password* | No | Password for network connection authentication | No | None |
| *proxy\_server* | No | Name of a proxy server if proxy is to be used | No | None |
| *connection\_timeout* | No | http(s) connection timeout value, in milliseconds | No | None |
| *read\_timeout* | No | Read timeout value, in milliseconds | No | None |
| *min\_threads* | No | The minimum number of threads created to process http requests | No | 1 |
| *max\_threads* | 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 |

## HTTPSenderModule

The HTTPSenderModule provides a mechanism for the workflows to make a HTTP(S) requests.

The module opens a http connection to target url. The workflows can then use the sendMessage node to make a request using this module.

The module can be used in two different ways:

1. all reqests are sent to the same URL in the same way. Please see description below for how to configure the url and method
2. the configuration for where to send the request is read from the service order registy and if this option is used no URL or method are needed to be configured for the module

The only necessary items to configure are:

* url The target URL for the HTTP(S) connection
* method The method to use to send the http message. Can be POST, GET, DELETE, PUT. Default is GET

In order to make a https connetion the following items must be configured:

* keystore The location of the keystore file necessary for HTTPS client authentication (optional)
* keypass The password for the public certificate/private key pair, necessary for HTTPS client
* authentication (optional)
* storepass The password to access the keystore file, necessary for HTTPS client authentication (optional)

**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 can be accessed through a proxy server then the following parameters are necessary:

* proxy\_server: Name of a proxy server (optional)
* proxy\_port: Port of a proxy server (optional)

**NOTE** 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.

You can configure multiple HTTPSenderModules, each must be given a unique name.

Retry attempts to process the request after a certain time can also be configured.

* fault\_tolerant: If set to true then retry will be performed in case of communication problems. Default is false.
* retry\_attempts: The number of times to retry processing the request. Default value is 0 (retry forever).
* sleep\_time: The time interval between each retry. The default value is 30 sec (must be configured in milliseconds)

All the passwords must be encrypted.

The parameter read\_message\_from\_db determines the whether messages are to be persisted to the database. I.e. if set to true then in case of fail-over, the messages that the failed node tried to send, but failed, will be picked up by the cluster node performing the fail-over.

Content type can be specified using the parameter content\_type.

Table 6‑10 HTTPSenderModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| connect\_timeout | Yes | The connection timeout, in milliseconds | No | 10 seconds |
| read\_timeout | Yes | Reading timeout | No | 10 seconds |
| fault\_tolerant | Yes | Indicates whether it is fault tolerant or not | No | true |
| retry\_attempts | Yes | Maximum number of retries for each message in case of failure | No | 0 |
| read\_message\_from\_db | Yes | Determines the whether messages are to be persisted to the database. I.e. if set to true then in case of fail-over, the messages that the failed node tried to send, but failed, will be picked up by the cluster node performing the fail-over. | No | true |
| content\_type | No | The request content type information | No | text/xml |
| keystore | No | The location of the SSL keystore file necessary for HTTPS client authentication | No | 30 min |
| keypass | No | The password for the public certificate/private key pair, necessary for HTTPS client authentication | No | None |
| storepass | No | The password to access the keystore file, necessary for HTTPS | No | None |

**Example: HTTPSenderModule**

<Module>

<Name>http\_sender\_module</Name>

<Class-Name>

com.hp.ov.activator.mwfm.engine.module.HTTPSenderModule

</Class-Name>

<Param name="connect\_timeout" value="10000"/>

<Param name="read\_timeout" value="10000"/>

<Param name="fault\_tolerant" value="true"/>

<Param name="retry\_attempts" value="3"/>

<Param name="read\_message\_from\_db" value="true"/>

</Module>

## 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 the document *HPE 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 HPE 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 prefixed 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 document *HPE Service Activator System Integrator’s Overview*

Table 6‑11 JMSListenerModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| workflow | Yes | Name of the workflow that is run to process each received message. | Yes | None |
| jndi\_initial\_ context\_factory | Yes | The JNDI initial context factory | No | None |
| jndi\_url\_pkg\_prefixes | No | package prefixes to use when loading in URL context factories. For JBossMQ it is org.jboss.naming:org.jnp.int erfaces | No | None |
| jndi\_url | Yes | The URL to reach the JNDI service to lookup the JMS destination. The format will  <url prefix>://<ip>:<port> | No | None |
| connection\_factory\_name | Yes | JMS connection factory name | No | None |
| jms\_trans\_ mode | Yes | JMS transfer mode, "topic" or "queue" | No | None |
| jms\_destination | No | Name of the destination to listen at (queue or topic) | No | None |
| durable | No | Indicates if a durable subscription shall be created | No | false |
| client\_id | Yes if durable | Client id for durable topic subscription | No | None |
| unsubscribe | No | Specifies whether the durable subscription must be unsubscribed when the module is stopped | Yes | false |
| username | Yes | User identity to create JMS connection | No | None |
| password | Yes | Password to create JMS connection | No | None |
| retry\_interval | No | Interval to wait between attempts to connecto the JMS provider. Defined in milliseconds | No | 1000 |
| max\_retry | No | Number of retries to connecto to the JMS provider before giving up. If set to 0 the module will continue to try to reconnect. | No | 3 |
| recover | 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 |
| header | No | Specifies whether or not to put an XML header before each received message.  Possible values are true and false. | No | true |
| dtd\_root\_tag | 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 |
| dtd | 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 | Exchang e.dtd |
| min\_threads | No | The minimum number of threads to use for processing arriving messages. | No | 1 |
| max\_threds | No | The maximum number of threads to use for processing arriving messages. | No | 1 |
| max\_queue\_length | 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 |
| write\_message\_to | No | Specifies where to persist incoming JMS messages (“file”, “db”, or “data”). When using “data” the message is kept in memory; i.e. not persisted. | No | "db" |
| filter | No | Specifies the JMS Message Selector that is going to be applied to every message in the queue/topic | No | None |

## 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 *HPE 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

* The appendix “Java Message Service” in document *HPE Service Activator, System Integrator's Overview*.

Table 6‑12 JMSSenderModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| jndi\_initial\_ context\_fact ory | Yes | The JNDI initial context factory | No | None |
| jndi\_url\_pkg\_prefixes | No | package prefixes to use when loading in URL context factories. For JBossMQ it is org.jboss.naming:org.jnp.int erfaces | No | None |
| jndi\_url | Yes | The URL to reach the JNDI service to lookup the JMS destination. The format is <url prefix>://<ip>:<port> | No | None |
| connection\_factory\_name | Yes | JMS connection factory name | No | None |
| jms\_trans\_mode | Yes | JMS transfer mode, "topic" or "queue" | No | None |
| jms\_destination | No | Name of the destination to listen at (queue or topic) | No | None |
| username | Yes | User identity to create JMS connection | No | None |
| password | Yes | Password to create JMS connection | No | None |
| retry\_interval | No | Interval to wait between attempts to connecto the JMS provider. Defined in milliseconds | No | 1000 |
| max\_retry | 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 HPE Service Activator, one module is automatically configured. If you change the database then you will need to manually update the datasoure parameteres. 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 6‑13 JNDIDatabaseModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *datasource\_*  *name* | Yes | The name of the datasource in JBoss which is used | No | None |

Example: 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 continously by using the distribution module to distribute jobs among active cluster nodes.
* Update the running modules and the resource manager whenver 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 supspended 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.

Table 6‑14 KeepAlive Module Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| *keep\_alive\_time* | 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 |
| *take\_over\_time* | No | Configurable time after which the KeepAlive module takes over a failed cluster node.  Generally, this should be twice the value of *keep\_alive\_time*. | Yes | 30000 |
| *job\_startup\_retry\_count* | 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 |
| *job\_startup\_retry\_interval* | No | Configurable interval time in milliseconds between retry attempts to start a job. | Yes | 10000 |
| *update\_heartbeat* | 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. If this parameter is set to false monitoring of other cluster node will not be done. | No | True |
| *monitor\_wait\_interval* | No | Configurable interval time in milliseconds that the KeepAlive module will wait from the start of its initiation or after loosing the database connectivity, before it will start monitoring of other cluster nodes. | No | 30000 |
| *db\_poll\_interval* | 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 |
| *retrieve\_jobs\_buffer\_size* | 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. |
| *configure\_virtual\_ip* | Yes | Indicates if the Keepalive module during startup must must try to set the configured virtual ip if any. | No | None |
| *auto\_virtual\_ip\_takeover* | 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 |
| *virtual\_ip\_ping\_timeout* | 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 miliseconds. | No | 10000 milliseconds |

Example: 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 HPE Service Activator username. The name of this attribute is configurable.
* User entries also contain the HPE Service Activator password for the user. The name of the password attribute is ‘userPassword’.
* Similarly, HPE 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 HPE 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 HPE Service Activator must also be created through the User Management Interface. Only users which have this configuration are able to login to HPE Service Activator. During installation of HPE 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 document *HPE Service Activator - System Integrator’s Overview*
* “Configuring Authentication or Authorization” in the document *HPE Service Activator—Developing Plug-Ins and Compound Tasks*
* “Setting Roles” on page 28

Table 6‑15 LDAPAuthModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| mwfm\_remote\_url | 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 HPE Service Activator | No | None |
| systemUserDN | 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 |
| systemUserFilter | No | Name of the attribute which contains the user name to login to HPE 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 |
| 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 | true |
| customErrorsFile | No | if set to true the module will search for $ACTIVATOR\_ETC/config/auth\_errors.properties file in order to replace certain LDAP errors to user defined ones. | No | false |
| check\_locally\_systemuser | No | if set to true, the system user will be authenticated without asking to LDAP server. | No | false |
| disable\_inactive\_users\_time | No | The number of days since the last time a user was successfully logged on until he has to be disabled. Defaults to 0, this meaning that no automatic disabling will be done. | No | 0 |

Regarding *customErrorsFile* file format, it can be consulted with examples in the file $ACTIVATOR\_ETC/config/auth\_errors\_example.properties.

Note that, in case *cutomErrorsFile* is true, and no replacement for a certain message is found, the UI will show the LDAP message. If that wants to be avoided is wise to set a “.\*” pattern as the lower priority pattern to show a generic error message.

Example: LDAPAuthModule

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\_hostname\_1" value="localhost" />

<Param name="ldap\_port\_1" value="389" />

<Param name="ldap\_active\_1" value="true" />

<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 HPE Service Activator must also be created through the User Management Interface. Only users which have this configuration are able to login to HPE Service Activator. During installation of HPE 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 document *HPE Service Activator - System Integrator’s Overview*
* “Configuring Authentication or Authorization” in the document *HPE Service Activator—Developing Plug-Ins and Compound Tasks*
* “Setting Roles” on page 28

Table 6‑16 LinuxAdvancedAuthModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| mwfm\_remote\_url | Yes | Used to indicate from where to get the remote Workflow Manager service | No | None |
| sleep\_time | No | The time between the internal role cache is cleared. | No | 30 min |
| eight\_char\_password | No | If password should be trucated to 8 characters before authentication is done | No | false |
| secure\_user\_name | No | Transform the user name to a valid value, cutting the user name from the first invalid character. | No | false |
| disable\_inactive\_users\_time | No | The number of days since the last time a user was successfully logged on until he has to be disabled. Defaults to 0, this meaning that no automatic disabling will be done. | No | 0 |

**Example: 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 6‑17 LoadFactorDistModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *load\_factor* | 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 |  |
| *dispatch\_local* | 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: 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 409
* “SolutionXMLLogModule” on page 400

Table 6‑18 LogSearchModule Parameters

| Parameter | Required | Description | Default |
| --- | --- | --- | --- |
| *commit\_interval* | No | The maximum amount of milliseconds between two consecutive commit operations to the log file index (default is 30 seconds). To optimize performance, the commit\_interval 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 |
| *commit\_max\_pending* | 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 |
| *field\_aliases* | 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 |
| *display\_first\_day\_of\_week* | 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 |
| *use\_24\_hour\_format* | No | Set this parameter to "true" (default) to use 24-hour format and to "false" to use 12-hour format on the UI. | “true” |
| *date\_format* | No | This parameter is used to configure the date format to be used on the UI | yyy-MM-dd |
| *max\_search\_results* | 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 435 for more information about Queue Hook modules.

Table 6‑19 MailHook Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Required | Description | Default |
| *smtp\_server* | Yes | The name of the SMTP server to be used for delivering the mail messages. | None |
| *mail\_account* | Yes | The mail account that is used to send the messages. It should be something of the form login@domain. | None |
| *domain* | Yes | The domain that is used when talking to the SMTP server. | None |
| *queue0, queue1... queueN* | Yes | Specify queue0 at a minimum. This parameter sets the name of the queue that will send an e‑mail notification to the addresses contained in the receiver0 text file. | None |
| *receiver0, receiver1... receiverN* | 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 |
| *html* | No | If set to false, messages are sent as plain text. Otherwise by default they are sent as HTML. | “true” |

## MailRequestModule

com.hp.ov.activator.mwfm.engine.module.notification.MailRequestModule

The module sends e-mail messages to different recipients using the SMTP protocol.

This module is used by the MailRequest node so it can be considered as synchronous.

**NOTE In general, the email is an asynchronous communication mechanism, but in this context it means that the workflow node execution will not keep waiting for the receipt confirmation from the email server. In addition, the message could be delivered to the email server, but it could be rejected afterwards due to several reasons, such as exceeding size limitation, incorrect attachment type, and so on.**

Usually email servers response with another email telling about the cause of the error. This email, however, is not managed by the module because the module acts only as a sender (so email reception protocols, like IMAP or POP are not supported).

There may be multiple sender modules configured within the Workflow Manager, using the module name as unique identifier in the workflow node as well as in the MWFM configuration file.

The module relies in the JavaMail library, so it supports the same configuration properties. The official website of the JavaMail is: <http://www.oracle.com/technetwork/java/javamail/index.html> while the related open source project is: <https://javaee.github.io/javamail/>.

**NOTE The supported properties can be different depending on the version. The properties for the last released version are detailed at: https://javaee.github.io/javamail/docs/api/com/sun/mail/smtp/package-summary.html.**

The existence of the “*mail.transport.auth.username*” and “*mail.transport.auth.password*” parameters in the module configuration implies their use for the creation of an authenticator during the server connection establishment. Otherwise the default instantiation without an authenticator is used.

Table 6‑20 MailRequestModule Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Required | Description | Default |
| *mail.transport.auth.username* | No | The name of the SMTP server to be used for delivering the mail messages | None |
| *mail.transport.auth.password* | Yes, if *mail.transport.auth.username* is specified | The mail account that is used to send the messages. It should be something of the following form: login@domain | None |
| *mail.smtp.host* | Yes | The URL or IP for the SMTP email server | None |
| *mail.smtp.port* | No | Default port for the SMTP email server | 25 |
| retry\_count | No | Number of times to retry processing the external request | 3 |
| retry\_interval | No | The interval in milliseconds to wait for next retry | 10000 |
| min\_threads | No | The minimum number of threads to manage external requests | 1 |
| max\_threads | No | The maximum number of threads to manage external requests | 3 |
| … | No | Any other property supported by the library: https://javaee.github.io/javamail/docs/api/com/sun/mail/smtp/package-summary.html | None |

**NOTE** The files to be attached must be located and be accesible in the *$ACTIVATOR\_VAR\_MAIL* folder. The module will not delete the file after the sending or after cleaning the folder. Users must manage the folder content, which implies the creation and the deletion of the files.

Due to a security restriction, it is not possible to use any other folder for attachments, avoiding to send a private or system configuration file by mistake.

## MailSenderModule

com.hp.ov.activator.mwfm.engine.module.notification.MailSenderModule

The module sends e-mail messages to different recipients using the SMTP protocol.

This module is used by the MailSend node so it can be considered as asynchronous.

**NOTE In general, the email is an asynchronous communication mechanism, but in this context it means that the workflow node execution will not keep waiting for the receipt confirmation from the email server. In addition, the message could be delivered to the email server, but it could be rejected afterwards due to several reasons, such as exceeding size limitation, incorrect attachment type, and so on.**

Usually email servers response with another email telling about the cause of the error. This email, however, is not managed by the module because the module acts only as a sender (so email reception protocols, like IMAP or POP are not supported).

There may be multiple sender modules configured within the Workflow Manager, using the module name as unique identifier in the workflow node as well as in the MWFM configuration file.

The module relies in the JavaMail library so it supports the same configuration properties. The official website of the JavaMail is: <http://www.oracle.com/technetwork/java/javamail/index.html> while the related open source project is: <https://javaee.github.io/javamail/>.

**NOTE The supported properties can be different depending on the version. The properties for the last released version are detailed at: https://javaee.github.io/javamail/docs/api/com/sun/mail/smtp/package-summary.html.**

The existence of the “*mail.transport.auth.username*” and “*mail.transport.auth.password*” parameters in the module configuration implies their use for the creation of an authenticator during the server connection establishment. Otherwise the default instantiation without an authenticator is used.

Table 6‑21 MailSenderModule Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Required | Description | Default |
| *mail.transport.auth.username* | No | The name of the SMTP server to be used for delivering the mail messages | None |
| *mail.transport.auth.password* | Yes, if *mail.transport.auth.username* is specified | The mail account that is used to send the messages. It should be something of the following form: login@domain | None |
| *mail.smtp.host* | Yes | The URL or IP for the SMTP email server | None |
| *mail.smtp.port* | No | Default port for the SMTP email server | 25 |
| retry\_attempts | No | Number of times to retry processing the external request | 10000 |
| sleep\_time | No | The interval in milliseconds to wait for next retry | 30000 |
| … | No | Any other property supported by the library: https://javaee.github.io/javamail/docs/api/com/sun/mail/smtp/package-summary.html | None |

**NOTE** The files to be attached must be located and be accesible in the *$ACTIVATOR\_VAR\_MAIL* folder. The module will not delete the file after the sending or after cleaning the folder. Users must manage the folder content, which implies the creation and the deletion of the files.

Due to a security restriction, it is not possible to use any other folder for attachments, avoiding to send a private or system configuration file by mistake.

## 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" in the document *HPE Service Activator - Introduction and Overview* for details on how you can use data collected.

Table 6‑22 Monitor Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Required | Description | Default |
| *database\_module* | Yes | The database module to use for writing audit records. If the database module is not specified, “db” is used. | “db” |
| *quantum* | 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 |
| *store\_statistics* | No | Indicates whether the Workflow Manager statistics is stored. | “false” |

Example: Monitor

<Module>

<Name>Monitor</Name>

<Class-Name>

com.hp.ov.activator.mwfm.engine.module.monitor.Monitor

</Class-Name>

<Param name="quantum" value="10"/>

</Module>

## NNMRequestModule

com.hp.ov.activator.mwfm.engine.module.nnmrequest.NNMRequestModule

NNMRequestModule enables workflows to execute request on Micro Focus NNMi\*. The workflows must use the nodes specially developed for interacting with Micro Focus NNMi\*.

You can configure the NNMRequestModule to access one Micro Focus NNMi\* server.

If you choose to configure multiple Micro Focus NNMi\* modules, give each module a unique name.

**NOTE** The password used to establish the connection can be encrypted. Use the crypt utiity to encrypt the password.

The module supports Micro Focus NNMi\* v10.

Table 6‑23 NNMRequestModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| nnm\_username | Yes | The Micro Focus NNMi\* server user name for authenntication. | No | None |
| nnm\_password | Yes | The Micro Focus NNMi\* server paasword for authenntication. | No | None |
| nnm\_pass\_is\_en crypted | No | Specifies if the provided nnm\_password is encrypted. | No | false |
| nnm\_hostname | Yes | The host name of the Micro Focus NNMi\* server. | No | None |
| nnm\_protocol | No | The protocol to be used for the connection to the Micro Focus NNMi\* server (either HTTP or HTTPS). | No | HTTP |
| nnm\_port | No | The Micro Focus NNMi\* server port. | No | 80 for HTTP and 443 for HTTPS |
| nnm\_keystore | No | Path to the keystore containing the Micro Focus NNMi\* certificate (used on HTTPS connections). | No | $ACTIVATOR\_ETC/config/mwfmSSL.keystore |
| nnm\_keystore\_p ass | No | The password for accessing and recovering keys from the KeyStore. | No | changeit |
| queue\_class | No | This can be set to the  com.hp.ov.activator.mwfm. module.WeightedEngineQueue, com.hp.ov.activator.mwfm. module.SimpleEngineQueue, or com.hp.ov.activator.mwfm.module.PriorityEngineQueue  The WeightedEngineQueue use the PRIORITY 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.  The SimpleEngineQueue will not do any prioritization of activation requests. They will be processed in FIFO order.  The PriorityEngineQueue uses the PRIORITY 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. | No | com.hp.ov.activator.mwfm.engine.module.WeightedEngineQueue |
| queue\_name | No |  | No | external\_request\_queue |
| 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 223.

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” for details about setting these parameters when a message is sent.

This module does not support re-configuration.

Table 6‑24 OVOMessageModule Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Required | Description | Default |
| *opcmsgcommand* | Yes | Used to specify the path to the opcmsg command. | None |
| *application* | No | Used to specify a default application name for messages sent by this module. The caller of the sendMessage() method can override the caller of the application. | HPE Service Activator |
| *msg\_grp* | No | Used to specify a default message group for messages sent by this module. The caller of the sendMessage() method can override the msg\_grp. | Misc |

**Example: 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 moudle 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 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 6‑25 QueueDistModule Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Required | Description | Default |
| *dispatch\_local* | 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 |

**Example: 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.queuing.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 379

Example: QueueModule

<Module>

<Name>queue\_module</Name>

<Class-Name>

com.hp.ov.activator.mwfm.engine.module.queuing.QueueDBPersistenceModule

</Class-Name>

</Module>

## QueueModule

com.hp.ov.activator.mwfm.engine.module.queuing.QueueModule

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 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 378

Example: QueueModule

<Module>

<Name>queue\_module</Name>

<Class-Name>

com.hp.ov.activator.mwfm.engine.module.queuing.QueueModule

</Class-Name>

</Module>

## QueueNotificationModule

com.hp.ov.activator.mwfm.engine.module.notification.QueueNotificationModule

Sender Module that works as message dispatcher to external systems through another Sender Modules or through customized workflows.

Supports queues and can send the same message/notification to a number of external systems which can be configured in the module by defining different sender modules or workflows.

Supports master/slave concept to ensure this is working across cluster nodes.

Persists the message before starting to send the message/notification to answer quickly back to the calling system no matter if the external systems are online or not and to support recovery in case HPE Service Activator crashes.

The name of the module must be queue\_notification\_module.

Table 6‑26 QueueNotificationModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| queue\_name\_N | For every configured queue. At least one queue is required. | Name of n-th queue, it must be unique among all the queues configured in the module. At least one queue is required, so queue\_name\_0 must always be configured. If more than one queue is needed then proceed with the next sequence number (queue\_name\_1, queue\_name\_2). | true | N/A |
| queue\_type\_N | true | Type of the n-th queue, it does not need to be unique. Incoming messages can be associated to one (or more) queue type and they will be sent through all the queues with the given queue type. | true | N/A |
| queue\_module\_N | true if queue\_workflow\_N is not specified | Name of the sender module through which all the messages buffered in the queue will be sent. It must exist a sender module configured with this name. This parameter is required for every queue configured in the module unless queue\_workflow\_N has been specified. If this parameter is defined then queue\_workflow\_N cannot be defined for the same queue. | true | N/A |
| queue\_workflow\_N | true if queue\_module\_N is not specified | Name of the workflow through which all the messages buffered in the queue will be sent. There must exist a workflow with this name in HPE Service Activator.This parameter is required for every queue configured in the module unless queue\_module\_N has been specified. If this parameter is defined then queue\_module\_N cannot be defined for the same queue. |  |  |
| queue\_customization\_class\_N | false | Specific customization class for this particular queue. If specified, then this one takes precedence over any other specified in customization\_class and this one will be applied to this queue. If not specified, then the one specified in customization\_class will be applied. If customization\_class is not specified either then no customization class will be applied for this queue. | true | N/A |
| retry\_interval\_N | false | Time to wait, specified in seconds, until a message is sent again in case of failure. Intervals may be different depending on the number of failed tries. E.g. "1,2,3" means that if the message cannot be sent for the first time the next retry must take place in 1 second, if it also fails wait 2 seconds and if this one also fails wait 3 seconds for the next retries until the value configured in 'retry\_count\_N' is reached. | true | 0 seconds (retry as soon as possible) |
| retry\_count\_N | false | Maximum number of times to retry sending messages in case of failure. | true | 0 (no retry) |
| persist\_failed\_messages\_N | false | Indicates whether messages that definitely cannot be sent must be persisted in database or not. If not persisted then they will be lost. | true | false (do not persist failed messages) |
| remote\_connectivity\_ issues\_wait\_time\_N | false | Number of seconds to wait in case of critical issues in the remote system. | true | 60 seconds |
| required\_queue\_types | false | Comma separated list of queue types. Incoming messages will necessarily be sent through these queues. If any other queue type is particularly specified for each incoming message then it will be sent through that one, too (avoiding repetitions). | true | None |
| default\_queue\_types | false | Comma separated list of queue types. Incoming messages with no queue type specified will be sent by default through the queues with the queue types in this list. If 'required\_queue\_types' is also configured then they list of queue types to use will be the result of concatenating 'default\_queue\_types' and 'required\_queue\_types' (avoiding repetitions), so incoming messages with no queue type specified will be sent through the queues with the queue types in the resulting list. | true | None |
| customization\_class | false | Configure this class to filter incoming messages (i.e. discard undesired messages) and translate them into a number of resulting messages meaningful to the external system to which they will be sent. If configured, this customization class will be applied to every queue except for those where a particular customization class is defined as well. Every class configured here must extend the abstract class com.hp.ov.activator.mwfm.engine.module.notification.QueueNotificationHook. | true | None |

Example: QueueNotificationModule

<Module>

<Name>queue\_notification\_module</Name>

<Class-Name>

com.hp.ov.activator.mwfm.engine.module.notification.QueueNotificationModule

</Class-Name>

<Param name="queue\_name\_0" value="http\_queue"/>

<Param name="queue\_type\_0" value="http"/>

<Param name="queue\_module\_0" value="http\_sender\_module"/>

<Param name="retry\_interval\_0" value="1,2,3"/>

<Param name="retry\_count\_0" value="5"/>

<Param name="queue\_name\_1" value="snmp\_queue"/>

<Param name="queue\_type\_1" value="snmp"/>

<Param name="queue\_module\_1" value="snmp\_sender\_module"/>

<Param name="retry\_interval\_1" value="1,5,10"/>

<Param name="retry\_count\_1" value="3"/>

<Param name="queue\_name\_2" value="snmp\_mirror\_queue"/>

<Param name="queue\_type\_2" value="snmp"/>

<Param name="queue\_module\_2" value="snmp\_mirror\_sender\_module"/>

<Param name="retry\_interval\_2" value="10,20"/>

<Param name="retry\_count\_2" value="2"/>

<Param name="default\_queue\_types" value="snmp"/>

<Param name="required\_queue\_types" value="http, snmp"/>

</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: 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 6‑27 RoundRobinDistModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *dispatch\_local* | 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 221
* “ModifyScheduledJob” on page 176
* “QueryScheduledJob” on page 196
* “DeleteScheduledJob” on page 107

Table 6‑28 SchedulerModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *max\_threads* | Yes | The maximum amount of threads in the pool used for launching the jobs at the specified time. | Yes | None |
| *min\_threads* | 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 max\_threads limit. Dynamic threads expire after 10 seconds of inactivity, or after the time specified in idle\_thread\_keep\_alive. | Yes | max\_threads |
| *idle\_thread*  *\_keep\_alive* | No | The amount of time (in seconds) for which an idle dynamic thread exists before it expires and is deleted (see the min\_threads).  This only applies to dynamic threads. | Yes | 10 |

Example: 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 HPE 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 6‑29 SelfMonitoringModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| *poll\_intrval* | No | The polling interval at which the parameters are verified.This is defined in microseconds. | No | 10000 |
| *threshold\_percent\_heap\_size* | 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 |
| *threshold\_perc ent\_maxworkli stlength* | 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 |
| *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 seperated list which indicat which time series should be save in the RRD database maintained 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: 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>

## ServiceOrderRegistryModule

com.hp.ov.activator.mwfm.engine.module.sor.ServiceOrderRegistryModule

Manages service order requests recived through the North Bound Interface in HPE Service Activator. The North Boud Interface provided with HPE Service Activator allows managing incoming service order messages. Each incoming SOR message will be persisted and processed in HPE Service Activator and eventually send a response back to the external system. The North Bound Interface grants that each message will never be processed twice even though the same SOR message can be received multimple times. A response message will be sent back to the external system if this is requested in the incoming SOR message. The North Bound Interface allows sending response messages multiple times in case of error.

The name of this module must be service\_order\_registry.

SOR messages will walk through the next steps until they are fully processed:

Query Message request is related to a query (i.e. read) operation

Received Message request has been received but has not been processed yet

Running Message request is being processed

Responding Response message is being sent to the External system

Finished Message request has been successfully processed

Gave Up Message request has been processed but response message cannot be sent back and has given up

Incomplete Message request has finished in an incomplete state

Gave Up Incomplete Message request has finished in an incomplete state and response message cannot be sent back

External For future use

Additional fields can be added for SOR messages. These customized fields will be translated into extra columns in the corresponding database table. HPE Service Activator provides a tool called SorAlterTable, accessible from command line in this location:

$ACTIVATOR\_HOME/bin

which must receive as parameter the path to an XML fiel that contains the specification of the new columns. This XML file must fulfill $ACTIVATOR\_ETC/config/sor.xsd.

The SOR UI is prepared to handle customized extra columns as described in HPE Service Activator User Guide.

Each SOR message is represented by an entry in the database table service\_order\_registry. This table is partitioned by the date field reception\_date. The partition range can be defined in the module configuration. The module will then manage partitions automatically. Due to this, if Oracle database is being used, the database user must have enough privileges to manage partitions. This can be achieved by running the next SQL commands:

grant redefine any table to ${role};

grant EXECUTE ON DBMS\_REDEFINITION to ${role};

where ${role} must be replaced by the user role or the username. If the database user does not have enough privileges, any partition management operation will end up in the following error:

PLS-00201: identifier ‘DBMS\_REDEFINITION’ must be declared.

**Note**: The module never deletes partitions. This task, if desired, must be performed manually in the database.

It is granted by the module that partition ranges will be created based on meaningful periods of time. For example, if the configured partition interval is set to MONTHLY, each partition will end the first day of the next month. The exception here (in Oracle databases) is the partition interval BIWEEKLY, which will end up creating partitions every 15 days, no matter if the resulting date is meaningful or not.

Table 6‑30 ServiceOrderRegistryModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| *time\_window* | No | Delete every message received a number of hours before the current system date. Must be specified in hours. Defaults to 1 hour. | No | 1 hour |
| *sleep\_interval* | No | Period of time to wait until the next group of messages in the specified time window is deleted. Must be specified in hours. The value 0 (default) means that no cleening up will be performed. | No | 0 hours |
| *partition\_interval* | No | The time interval required for each partition that is going to be created by the module. Accepts the values BIWEEKLY, MONTHLY (default), BIMONTHLY, ONCE\_A\_QUARTER, ANNUAL, BIANNUAL and TRIANNUAL. | No | MONTHLY |

## ServiceOrderRegistryMonitoringModule

com.hp.ov.activator.mwfm.engine.module.sor.ServiceOrderRegistryRrdModule

Monitors the service order registries managed by the service\_order\_registry module, collecting information that is saved in time series in a Round-Robin database. This information can be seen in the User Interface. Data regarding maximum, minimum and average duration of service orders, how many of them have been received and how many are in the different allowed states will be saved.

Data related to maximum, minimum and average duration will be collected separately for quey, write and other kind of operations. Service order registries related to query operations will be those in state *query*; service order registries related to write operations will be those whose operation name fits with any of the configured values (see the table below); the rest of the service order registries will be considered as other kind of operations.

Table 6‑31 ServiceOrderRegistryMonitoringModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| *granularity* | No | A comma seperated list indicating which time series should be saved in the RRD database maintained and handled by the module. Every granularity must be multiple of the lowest granularity configured. Values are defined in minutes. The minimum value accepted is 1 minute. | No | 1,5,30,240,1440,10080 |
| *samples* | No | The number of samples which are saved for each time serie. | No | 360 |
| *write\_operations* | No | A comma separated list indicating the operation names (case sensitive) from the service order registries that will be considered *write* operations. | No | modify,create,delete, POST,DELETE |

Example: ServiceOrderRegistryMonitoringModule

<Module>

<Name>service\_order\_registry\_monitoring</Name>

<Class-Name>

com.hp.ov.activator.mwfm.engine.module.sor.ServiceOrderRegistryRrdModule

</Class-Name>

<Param name="granularities" value="1,5,30,240,1440,10080"/>

<Param name="samples" value="360"/>

<Param name="write\_operations" value="modify,create,delete,POST,DELETE"/>

</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 6‑32 SNMPSenderModule Parameters

| Parameter | Required | Description | Default |
| --- | --- | --- | --- |
| *host* | Yes | host name of the SNMP manager. | None |
| *port* | No | Port in which the SNMP manager is listening on | 162 |
| *engine\_id* | Yes, if SNMP version is 3 | Engine Id of the agent. | None |
| *username* | Yes, if SNMP version is 3 | User name to be used. Used for version 3 traps. | None |
| *password* | 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 |
| *auth\_protocol* | No | Authorization protocol to be used. It can either be MD5/SHA1. Used for version 3 traps. | MD5 |
| *community* | No | Community to be used while sending the SNMP trap. Used for version 2 traps. | public |
| *privacy\_protocol* | No | Protocol to be used for privacy. The value can either be DES/AES. Used for version 3 traps. | DES |
| *privacy\_pwd* | No | Privacy password to be used. | In case the password parameter has been defined then the default value of privacy\_pwd will be the same as that of the password parameter. |
| *snmp\_version* | No | The snmp version to be used. Valid values are 2 and 3. | 3 |

Example: 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="caLbA9g6hlnl/lQXeTL/mQ=="/>

<Param name="community" value="public"/>

<Param name="auth\_protocol" value="MD5"/>

<Param name="privacy\_protocol" value="DES"/>

<Param name="privacy\_pwd" value="caLbA9g6hlnl/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.

**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 212 for information about removing a file after a workflow has finished processing it.
* “SendMessage” on page 227 and “SocketSenderModule” on page 398 for information about sending messages back to a waiting program.
* “WorkManagerModule” on page 408 for information about workflow prioritization.

Table 6‑33 SocketListenerModule Parameters

| Parameter | Required | Description | Default |
| --- | --- | --- | --- |
| *workflow* | Yes | The workflow to be started upon receipt of a message. A parameter with the name message\_url (containing the file where the message is stored or the message id refering to a row in the table where the message is stored) is passed through to the new workflow instance just created. | None |
| *port* | Yes | The port on which the module listens. | None |
| *recover* | No | If set to “true”, on start-up, all existing messages in the directory received\_messages or unprocessed messages in the DATABASE MESSAGE 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” |
| *directory* | No | The directory (*$ACTIVATOR\_VAR*/) where messages are held until processing. | *$ACTIVATOR\_VAR*/received\_ messages |
| *save\_messages* | No | Enables message logging to the *$ACTIVATOR\_VAR*/received\_messages\_log directory. | “false” |
| *header* | No | Specifies whether the XML header should be put before the message being received or not. Possible values are true and false. | “true” |
| *dtd\_root\_tag* | No | If you specify this parameter, the header of the XML document is set to point to it. If it is not specified and a DTD is present, its value is msg. | msg |
| *dtd* | No | Specifies the document type definition that should be used for validating the XML message. You can specify the absolute location of the DTD file, or a path relative to  *$ACTIVATOR\_ETC*/config. If you do not want to specify a DTD, set the value to "NO\_DTD". | exchange.dtd |
| *keystore* | 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 |
| *keystore\_password* | No | The password to use to open the keystore. | None |
| *clientauth* | 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” |
| *min\_threads* | No | The minimum number of threads to maintain for processing arriving messages. | 1 |
| *max\_threads* | No | The maximun number of threads to maintain for processing arriving messages. | 3 |
| *write\_message\_to* | 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.

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 227 for more information about sending messages from a workflow to a waiting program.

Table 6‑34 SocketSenderModule Parameters

| Parameter | Required | Description | Default |
| --- | --- | --- | --- |
| *Host* | Yes | The host to send the messages to. | None |
| *port* | Yes | The port in the host to send the messages to. | None |
| *fault\_tolerant* | No | If “true”, messages that cannot be sent for any reason are saved and retried after a few seconds (see *sleep\_time*). | “false” |
| *sleep\_time* | No | Use this parameter to specify a retry interval when fault-tolerant is set to true. Specify the number of seconds to wait before retry. | 30 |
| *keystore* | 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 |
| *keystore\_password* | No | The password to use to open the keystore. | None |
| *pending\_message\_directory* | No | Sets the pending messages directory. If you specify this parameter, you will override the default directory *$ACTIVATOR\_VAR*/pending\_messages. This is useful if you have multiple SocketSenderModules configured. | None |
| *read\_data\_from\_db* | 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 Workflow Manager 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 165

Table 6‑35 SolutionXMLLogModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| *log\_max\_entries* | 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 |
| *log\_level* | No | The following log\_level 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 |
| *log\_allow\_statistics* | No | A value of “true” allows the logging of statistical data. This is distinct from the level of logging that is chosen. | No | “false” |
| *log\_directory* | Yes | The location where the log files will be stored under the hostname folder. | No | VAR\_HOME/log |
| *solution\_name* | Yes | Specifies the name of the solution that is used to generate log files. | No | None |
| *log\_max\_files* | 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 |
| *index* | 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: 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 SyncModule 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 SyncModule 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 the document *HPE Service Activator System Integrator's Overview HPE Service ActivatorDeveloping Plug-Ins and Compound Tasks*
* “Setting Roles” on page 28
* “AskFor” on page 81

Table 6‑36 SyncModule Parameters

| Parameter | Required | Description | Reconfigurable | Default |
| --- | --- | --- | --- | --- |
| *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: 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>

## 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 6‑37 UsageMonitoringModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *collection\_interval* | No | The interval (in seconds) between two consecutive usage data collection events.  The minimun allowed value for this parameter is 1 minute (60 seconds). | No | 3600 |
| *usage\_thredshold* | 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: UsageMonitoringModule**

This example configures the UsageMonitoringModule.

<Module>

<Name>usage\_monitoring\_module</Name>

<Class-Name>

com.hp.ov.activator.mwfm.engine.module.UsageMonitoringModule

</Class-Name>

<Param name="collection\_interval" value="3600"/>

<Param name="usage\_thredshold" 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 HPE Service Activator must also be created through the User Management Interface. Only users which have this configuration are able to login to HPE Service Activator. During installation of HPE 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 HPE Service Activator to perform this kind of authentication. See the discussion “Update Local Security Policies” of the HPE Service Activator Installation Guide for Windows.

**See Also**

* “Roles, Privileges, and Authentication” in the document *HPE Service Activator - System Integrator's Overview*
* “Configuring Authentication or Authorization” in the document *HPE Service Activator-Developing Plug-Ins and Compound Tasks*
* “Setting Roles” on page 28

Table 6‑38 WindowsAdvancedAuthModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *mwfm\_remote\_url* | Yes | Used to indicate from where to get the remote Workflow Manager service | No | None |
| *sleep\_time* | No | The time between the internal role cache is cleared. | No | 30 min |
| *eight\_char\_password* | No | If password should be trucated to 8 characters before authentication is done | No | false |
| *secure\_user\_name* | No | Transform the user name to a valid value, cutting the user name from the first invalid character. | No | false |
| disable\_inactive\_users\_time | No | The number of days since the last time a user was successfully logged on until he has to be disabled. Defaults to 0, this meaning that no automatic disabling will be done. | No | 0 |

Example: 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 6‑39 WorkManager Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| queue\_class | No | Indicates which class implements the queue for ordering work items. | No | com.hp.ov. activator.mwfm. engine.module. WeightedEngineQueue |

**Example: 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”.

Table 6‑40 XMLLogModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *log\_max\_entries* | 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 |
| *log\_max\_files* | 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 |
| *log\_level* | No | The following log\_level 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 |
| *log\_allow\_statistics* | No | A value of “true” allows the logging of statistical data. This is distinct from the level of logging that is chosen. | No | “false” |
| *log\_max\_files* | 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 |
| *log\_directory* | Yes | The location where the log files will be stored under the hostname folder. | No | VAR\_HOME/log |
| *index* | 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: 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>

## WSModule

com.hp.ov.activator.mwfm.engine.module.WSModule

The Web Service module provides the connectivity with external endpoints using SOAP connections. The connection is delegated to the JBossWS.

To include a new module requires several previous steps:

* Get the WSDL file for the service:
  + All dependencies (like external XSD or linked WSDL) for this file must be resolved locally so it could be necessary to get these files and move manually to the local system (modifying the WSDL file in order to resolve the dependencies locally).
  + WSDL file must not contain the XML header: “<?xml version="1.0" encoding="UTF-8" standalone="yes"?>”.
* Generate client executing the utility: “wsconsume” that is located in $JBOSS\_HOME/bin. The command output will be a folder, named “output”, located in the directory where the command will be executed and that contains a set of JAVA files and classes for the client. The syntax for the command is:

/[$JBOSS\_HOME]/bin/wsconsume -b jaxb-bindings.xml -k service.wsdl

The file ‘jaxb-bindings.xml’ is located in the folder $ACTIVATOR\_ETC/config and will generate a set of serializable java classes.

* Compress the client files to a JAR file, using the command syntax:

jar –cf myService-1.0.0.jar –C output .

Note that “jar” utility is distributed in the JDK and located in the folder: $JAVA\_HOME/bin . Besides the name “myService-1.0.0.jar” is only an example but it has to be replaced by other appropriated name.

* The “output” folder is not longer required and must be deleted.
* Move the generated jar file to the folder: $JBOSS\_EAR\_LIB.
* Generate the descriptor for the module. The command syntax is:

$ACTIVATOR\_BIN/generateWSDescriptor myService-1.0.0.jar myService-1.0.0.xml

The first argument for the command is the previously generated JAR file. And the second argument is the name of the file to be generated. If the descriptor is going to be located in the filesystem, then must be in the folder: $ACTIVATOR\_ETC/config/ws

* Configure the module in the mwfm.xml

Table 6‑41 WSModule Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Required | Description | Reconfigurable | Default |
| *url* | Yes | The URL for the external system.  It can contains a list of URLs, using a semicolon as separator. | Yes |  |
| *wsdl* | Yes | The location for the WSDL file that defines the message structure. If it’s in the filesystem, the URI must start with “file:/” | Yes |  |
| *descriptor* | Yes | Mandatory parameter that defines where is located the associated descriptor for the WS. It must start with the prefix: 'db:' or 'file:'. If it is saved in the database (because it was deployed with the Deployment Manager), then the parameter solution\_name is mandatory. If it located in the filesystem, then it can be locally resolved in two ways:  - if it contains only the filename then it will be in the folder: $ACTIVATOR\_ETC/config/ws  - or it can use an absolute path. | Yes |  |
| *solution\_name* | Yes (if descriptor is in the database) | Describes to what solution belongs. | Yes |  |
| *username* | No | Username used for request authentication. | Yes |  |
| *password* | No | Password used for request authentication. | Yes |  |
| *encrypted\_password* | No | Encrytped password for request authentication.  It is not compatible with the parameter “password”. | Yes |  |
| *initialize* | No | The connection is instantiated during the module initialization. The first request will be executed more fast but the service will start more slow. | Yes | false |
| *load\_balance* | No | If the load\_balance is true then a new created connection if the current ones are exhaust (looking for the next one of the URL list. Else it will use only the first URL. | Yes | false |
| *reuse\_connection* | No | If is configured to ‘false’ then a new endpoint implementation class will be instantiated per request. If it is set to ‘true’ then it will be reused (and the performance will be better). | No | false |
| *internal\_XYZ* | No | A set of additional parameters that are set to the JbossWS connection configuration. XYZ has to be replaced by the proper constant. In example, to configure the timeout (<https://cxf.apache.org/javadoc/latest/constant-values.html#org.apache.cxf.message.Message.CONNECTION_TIMEOUT>) the key must be: ‘internal\_javax.xml.ws.client.connectionTimeout’ and the value must be the desired timeout in millis. The SSL connections must be configured in the same way according with the Jboss official doc: <https://docs.jboss.org/author/display/JBWS/WS-Security#WS-Security-Client> | Yes |  |

Example 6-32 WSModule Code

<Module>

<Name>hpsa\_ws\_module</Name>

<Class-Name>com.hp.ov.activator.mwfm.engine.module.WSModule</Class-Name>

<Param name="descriptor" value="hpsaTestDescriptor.xml"/>

<Param name="wsdl"

value="file:/C:/HP/OpenView/ServiceActivator/etc/config/ws/hpsatest.wsdl"/>

<Param name="url"

value="http://localhost:8080/WSEndpointTest/WSEndpointTest"/>

<Param name="username" value="admin"/>

<Param name="password" value="mySecretPasswd"/>

</Module>

### WSDescriptor

The descriptor is an important part of the module and defines the metadata information for the service, like available operations/methods, input and output parameters for each operation, optional aliases for each parameter and canonical references to the Java objects.

It can be located in the filesystem or in the database (in this case, it must be deployed with the Deployment Manager). The file format is XML and must comply the XSD ‘wsDescriptor.xsd’.

Through the descriptor is possible to define an alias for any operation or parameter and use the friendly reference in the node WSRequest. What simplifies the node legibility&use and provides an abstraction layer between the technical implementation of the WS and the workflows. In example this could be the descriptor for the a queryLocation operation:

…

<operation>

<name>queryLocation</name>

<requestConfigurations>

<requestConfiguration name="default">

<requestObjectClassname type="com.hpe.cms.ws.Header">javax.xml.ws.Holder</requestObjectClassname>

<requestObjectClassname>com.hpe.cms.ws.QueryLocationBody</requestObjectClassname>

</requestConfiguration>

</requestConfigurations>

<invokerMethod>queryLocationFromInventory</invokerMethod>

<inputs>

<input requestConfiguration="default">

<parameter class="javax.xml.ws.Holder" type="com.hpe.cms.ws.Header" argumentOrder="0">

<alias>Header</alias>

<path>Header</path>

</parameter>

<parameter class="java.lang.String" argumentOrder="0">

<alias>HeaderVersion</alias>

<path>Header.version</path>

</parameter>

<parameter class="com.hpe.cms.ws.QueryLocationBody" argumentOrder="1">

<alias>QueryLocation</alias>

<path>QueryLocation</path>

</parameter>

<parameter class="com.hpe.cms.ws.LocationQuery" argumentOrder="1">

<path>QueryLocation.locationQuery</path>

</parameter>

<parameter class="java.lang.String" argumentOrder="1">

<alias>customeraddress</alias>

<path>QueryLocation.locationQuery.address</path>

</parameter>

…

</input>

</inputs>

<output>

<parameter>

<alias>firstreturnedcountry</alias>

<path>location#0.country</path>

</parameter>

…

In the workflow, it would be possible to reference the aliases in the WSRequest node:

<Process-Node>

<Name>WSRequest</Name>

<Action>

<Class-Name>com.hp.ov.activator.mwfm.component.builtin.WSRequest</Class-Name>

<Param name="module" value="hpsa\_ws\_module"/>

<Param name="operation" value="queryLocation"/>

<Param name="response\_mode" value="CLASSIC"/>

<Param name="input\_param0" value="customeraddress"/>

<Param name="input\_variable0" value="addressVar"/>

<Param name="output\_param0" value="firstreturnedcountry"/>

<Param name="output\_variable0" value="countryVar"/>

<Param name="input\_param1" value="Holder.version"/>

<Param name="input\_variable1" value="ws\_request\_versionVar"/>

</Action>

<Next-Node>WasPreviousNodeOK</Next-Node>

</Process-Node>

In the previous snippet, instead of to use the path of the input parameter ‘QueryLocation.locationQuery.address’, the alias ‘customeraddress’ is referenced.

It’s important to note that the path cannot be modified because it defines how to fill the WS request (how to nest the information in the request). For example:

<parameter class="java.lang.String" argumentOrder="1">

<alias>customeraddress</alias>

<path>QueryLocation.locationQuery.address</path>

</parameter>

This means that:

* The type of the parameter in Java is ‘java.lang.String’ and it is the second argument in the request.
* An alias, named ‘customeraddress’ can be referenced in the workflows. This alias can be specified by the delivery team, system integrators … and simplifies the workflow implementation.
* The path defines how to fill the request object. Internally it will assign the value to the Java object: this.getQueryLocation().getLocationQuery().setAddress(…). Note that ‘QueryLocation.locationQuery’ has been defined as a previous parameter so this is nesting the information. The structure of the generated XML request will be:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:hpenp="http://www.hpe.com/tnpNML/">

<soapenv:Header>

<hpenp:Header>

…

</hpenp:Header>

</soapenv:Header>

<soapenv:Body>

<hpenp:QueryLocation>

<LocationQuery>

<Address></Address>…

</LocationQuery>

</hpenp:QueryLocation>

</soapenv:Body>

</soapenv:Envelope>

It’s possible to use the path or the alias for the input/output parameters in the workflows. The first element of the result list (‘<path>location#0.country</path>’) is a valid example where the alias ‘firstreturnedcountry’ is referenced in the workflow. The returned value will be mapped out to the case packet variable: ‘countryVar’ (but if the returned list is empty, an exception will be thrown because at least one result is expected).

Inheritly it is possible to manage list of returned elements, using the separator char ‘#’, that indicates the position of the element in the list. For example, the previous path ‘location#0.country’ will correspond to the first position of the ‘location’ list (note that count starts in 0).

### Dependency management

In the previous versions of the HPE Service Activator the Java Virtual Machine included support for JAXB libraries natively. Since it has been removed in the JDK 11, some WS clients could require a migration to resolve properly these missing dependencies: including an external JAR file with the dependencies, reimplementing the client …

Note all clients (running as implementations of WSModule) share the same classloader so customers have to be careful and ensure the new dependencies JAR files are fully compatible with other implementations. In this sense if customers add two or more JAR files for supplying the same missing dependency could cause an error due to the duplication or cause an unexpected/random behavior.

### Fault and exception management

Please see the section *Fault Management* in the *GenericWS* documentation.

# Writing Custom Workflow Nodes

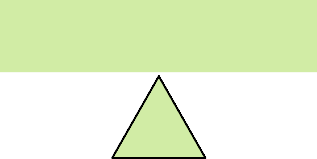
You may find it necessary to develop new workflow nodes to accomplish specific tasks in your business processes. This chapter provides conceptual information and instructions for writing new process nodes.

## 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



WFComponent

W

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | | | |
|  |  | | |  | | |  |
| FProcessNode | |  | WFRuleNode | |  | Handler | |



|  |  |
| --- | --- |
|  |  |
| ErrorHandler | |

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 | com.hp.ov.activator.mwfm.component.WFProcessNode |
| Rule nodes | com.hp.ov.activator.mwfm.component.WFRuleNode |
| Switch nodes | com.hp.ov.activator.mwfm.component.WFSwitch |
| Error handler | com.hp.ov.activator.mwfm.component.ErrorHandler |
| End handler | com.hp.ov.activator.mwfm.component.Handler |

**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 HPE 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: 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 WFConfigException

{

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 WFConfigException(

MessageFormat.format("Parameter {0} must be a variable.",

new Object[]{ObjectConstants.OPERAND+0}) );

}

varToSet = str;

}

attributes.addElement( str );

}

if (attributes.isEmpty()) {

throw new WFConfigException(

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: 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: 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 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" );

#### 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: 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 HPE 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 HPE Service Activator has be started. If you change your jar files, then you must stop and restart HPE 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 Workflow 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 sub-directories 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  icon 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: 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">

<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.

# 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

HPE 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 an 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 (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 SocketListenerModule will notifly 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 KeepAliveModule 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 KeepAliveModule 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 SchedulerModule tries to become the master schedulder 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 KeepAliveModule 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 SchedulerModule tries to become the master schedulder for the cluster.

void nodeLocked(ClusterNodeBean node)

#### nodeUnlocked Method

When the lock state of other cluster nodes change from locked to unlocked, the KeepAliveModule 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 KeepAliveModule 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 SchedulerModule 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 KeepAliveModule 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 taken over 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 KeepAliveModule 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 SyncModule,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 completionof the take over process, and the the KeepAliveModule 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 SyncModule 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 “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 create MasterSlaveState() 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 method 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 s 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), incase 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 HPE 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 verfies tath the user is allowed to use HPE 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.UserManagmentManger.

Almost all the methods in the UserManagmentManager (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 know by HPE Service Activator and the relation from roles to inventory tress, 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 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 newUserName, 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 “MailHook” on page 368) 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 “hook*N*” 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: QueueHook example

The source for the MailHook is provided as an example of a QueueHook. See the Java file *$ACTIVATOR\_OPT*/examples/modules/MailHook.java which is available then HPE Service Activator has been installed.

### 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 HPE 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 HPE Service Activator has be started. If you change your jar files, then you must stop and restart HPE Service Activator to pick-up the new implementation.

# 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: 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.

\*/

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: 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.

\*/

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));

}

}

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 HPE 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 HPE 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 HPE 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 HPE Service Activator. Note, however, that mwfmtool allows you to run scripts, which automates your routine tasks but requires some scripting or programming experience.

**NOTE** The tool is called mwfmtool.bat on 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...>

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 press **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 mwfmtool from a Remote Computer

If you use mwfmtool on the computer where HPE Service Activator is installed, then you can skip this section.

You can, however, use mwfmtool without installing HPE Service Activator. In this case, you would use mwfmtool to connect to a remote computer, on which HPE 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 9‑1 to your computer without HPE Service Activator. The column “Location” in Table 9‑1 contains the locations where the files can be found in a typical installation of HPE Service Activator.

For further instructions, see “Move mwfmtool to a Computer Running UNIX” or “Move mwfmtool to a Computer Running Microsoft Windows”.

Table 9‑1 mwfmtool Files

|  |  |
| --- | --- |
| File | Location |
| mwfmtoolusage.txt | $ACTIVATOR\_ETC/config |
| mwfmRB\_en.properties | $ACTIVATOR\_ETC/nls |
| activator\_utils.jar | $ACTIVATOR\_OPT/lib |
| mwfm.jar | $ACTIVATOR\_OPT/lib |

Move mwfmtool to a Computer Running UNIX

This section has the instructions on how you can move mwfmtool to a computer running UNIX. In these instructions, the computer which has HPE 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

/opt/mwfmtool/bin

/opt/mwfmtool/etc/config

/opt/mwfmtool/etc/nls

/opt/mwfmtool/lib

1. Locate the files listed in Figure B-1 in Computer A.
2. Copy the files mwfm.jar and activator\_utils.jar to the directory /opt/mwfmtool/lib
3. Copy the mwfmtoolusage.txt to the directory /opt/mwfmtool/etc/config
4. Copy the mwfmRB\_en.properties to the directory /opt/mwfmtool/etc/nls
5. Create a script called mwfmtool in the directory /opt/mwfmtool/bin. The content of the script must be the following

#!/bin/bash   
ACTIVATOR\_ETC=/opt/mwfmtool/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 \

-DMWFMTOOL\_ETC=$ACTIVATOR\_ETC \

com.hp.ov.activator.mwfm.client.mwfmtool “$@“

1. To start mwfmtool, follow the instructions in “Start mwfmtool”.

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 HPE 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

1. Locate the files listed in Figure B-1 in Computer A.
2. Copy the files mwfm.jar and activator\_utils.jar to the directory :\HP\OPenView\mwfmtool\lib
3. Copy the mwfmtoolusage.txt to the directory C:\HP\OPenView\mwfmtool\etc\config
4. Copy the file mwfmRB\_en.properties to the directory C:\HP\OPenView\mwfmtool\etc\nls
5. 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% \

-DMWFMTOOL\_ETC=%ACTIVATOR\_ETC% com.hp.ov.activator.mwfm.client.mwfmtool %\*

1. To start mwfmtool, follow the instructions in “Start mwfmtool”.

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 9‑2 Command Structure

|  |  |
| --- | --- |
| Parameter | Description |
| <cmd> | Command name. See Table B-3, which contains the command names. |
| [-host<hostname>] | Host name. By default, mwfmtool 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. -host \*.\*.\*.\* |
| [-port<port>] | Port name. If left out, mwfmtool assumes the default port of the Workflow Manager. In typical installations, it is 2000. If your HPE Service Activator is set up otherwise, provide here the port number used by the Workflow Manager, e.g. -port 2001 |
| [-user<username>[/<password>]] | The user name and the password you use to log on to HPE Service Activator when user name and password authentication is enabled, e.g.  -user aaa/\*\*\* In this example, “aaa” is the user name while the asterisks represent the password. |
| [-quiet] | This parameter allows you to control the output returned by the commands. It accepts either -verbose or -quiet as its values. Use  -verbose if you want your commands to  generate detailed information about their results. Use -quiet to turn off all output. |
| <...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. |

Table 9‑3 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 |
| 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 sesions and database pools. Specify the host name as it is stored in database table CLUSTERNODELIST from where this information must be retrieved. |
| 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 |
| GetMutexStatus | not available | Get information about mutex status (i.e. acquired and pending locks) in the Conflict Module. |
| 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 sesions and database pools. Specify the host name as it is stored in database table CLUSTERNODELIST from where this information must be retrieved. |
| GetValidRoles | not available | Gets the list of valid roles according to the Authentication module. |
| GetVersion | not available | Get the version of the HPE Service Activator application |
| GetWFMConfiguration | not available | Get a snapshot of the current WFM configuration. |
| 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. |
| 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 for more details. |
| ShowDatabaseMessages [-mrssageId  <mesageId>][-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! |
| 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> | not available | 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 |
| QueryRequestQueueJobs -queue  <queueName> [-serviceId  <serviceId>] [-orderId <orderId>] [-type <type>] [-state <state>]  [-maxRecords <max records>] | not available | Shows the jobs in a queue which match the query parameters |
| QueryRunningJobs [-serviceId  <serviceId>] [-orderId <orderId>] [-type <type>] [-state <state>]  [-maxRecords <maxRecords>] | not available | 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. |
| ScheduleJob  -time<timestamp> [repeatingPeriod<period>  -repeatingPeriodUnit <unit>  -repeatingEnd<end>  -repeatingType <type>]  [-description<description>] [-groupID<groupID>]  [-status<status>]  [[-start\_missed\_scheduled\_inst ances <true|false>]  [-noinput |  -messageFile<file>]  <workflowName> | not available | Schedules a workflow. The  -time argument tells the command at which time to schedule a job.  -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.  -repeatingType indicates whether the Scheduled Time calculated for reoccurring jobs on restarts is relative or absolute. Allowed types:  1-relative, 2-absolute  -groupID is used to add several jobs to a group. -description is a short description of a job.  -status marks the current status of a job.  -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.  -messageFile sets the message\_file case-packet variable to the given file. |
| GetScheduledJob  <jobID> | not available | Returns all attribute of a scheduled job. |
| DeleteScheduleJob <jobID> | not available | Deletes a scheduled job. |
| MakePrimarySite <site name> | not available | Makes the specified site as the primary site when HPE 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: 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: 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

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 predefied datasources delivered with HPE Service Activator 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 HPE Service Activator 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>